Robust plan execution via reconfiguration and replanning

Robust Plan Execution via Reconfiguration and Replanning

Enrico Scala*, Roberto Micalizio**, Pietro Torasso**
Dipartimento di Informatica
Università di Torino
*enrico.scala@anu.edu.au
**{micalizio, torasso}@di.unito.it

August 5, 2015

Abstract

Acting in real world may be a difficult task for an agent, either software or robotic, because unexpected contingencies may arise at any step of the execution. Previous approaches to robust plan execution consider propositional goals to be achieved and time constraints to be satisfied. However, realistic plans must obey to constraints on continuous/consumable resources, too.

To face the complexity in handling these resources, the paper proposes the notion of Multi Modality Action (MMA). The model allows to explicitly express the multiple execution modalities in which a given action can be executed; each execution modality models requirements/consequences on the involved consumable resources when that modality is selected. Relying on the MMA notion, the paper presents how the repair problem can be seen as a problem of reconfiguring actions modalities, and how it can be solved by exploiting a CSP encoding.

The MMAs are employed by a new continual planner, FLEX-RR, which, exploiting the synergy from the reconfiguration and a numeric planning mechanism can efficiently repair on the fly the plan keeping it rather stable. An empirical analysis performed on three numeric planning domains, confirms the large benefits of FLEX-RR in terms of competence, efficiency and stability of the repaired plan.

1 Introduction

Real-world scenarios are in general weakly predictable and highly dynamic. This means that unexpected contingencies may arise at any step of the execution. It is not thus surprising that, as also recently stated in [15], the problem of *robust plan execution* in real-world domains is a very challenging topic in AI planning, necessary for implementing planning mechanisms in autonomous systems.

A first methodology to cope with the problem consists in anticipating, at planning time, all the possible contingencies that might occur during the actual execution of the plan. The result of the planning phase is therefore a conditional plan [5, 18], i.e. a plan where alternative courses of actions are possible depending on certain conditions of the environment. During the execution phase, the agent, guided by its sensing actions, is able to select a feasible branch of its plan that leads to the goal. A similar methodology has been proposed in [6]; in this case, conditional plans are built to guarantee that the agent satisfies temporal constraints on the achievement of its goals.

These approaches are successful when it is possible to find a plan which anticipates all the contingencies at planning time (i.e., off-line). When such contingencies cannot be handled off-line, alternative on-line solutions are required. Many works [40, 9, 14, 11] have recently proposed to enhance robust plan execution via *plan repair* techniques. Basically, these works suggest to stop the plan execution as soon as some unexpected situation is encountered, and then they attempt to repair the broken plan either via replanning from scratch or via plan adaptation ([13]).

However, adapting a plan can be very hard in the general case ([32]), and even undecidable when the domain involves resources modeled as numeric state variables ([16]). To handle the repair problem in a more efficient fashion, this paper addresses the robust execution from a different perspective: to limit as much as possible the need of replanning, we propose a new characterization of the repair given in terms of a reconfiguration.

Similarly to the numeric extension of PDDL ([10]), we start from the observation that real-world scenarios include both reusable and consumable resources. Thus, a plan has not only to achieve a set of goals, but it has also to conform to strict constraints on the amount of resources to be used by the agent. For this reason, it is necessary to explicitly model the (expected) profile of resource consumption during the execution of any action. We thus model resources as numeric fluents, which can be explicitly mentioned in the preconditions and effects of the action templates[1].

We also observe that, in many real-world domains it is possible to identify actions performing the same task (i.e., obtaining the same effects), but requiring different configurations of the agent. These actions share the same qualitative objectives, while they differ in the way they are performed. Typically, different configurations have different resource profiles. We therefore propose to group these subsets of similar actions by introducing the notion of *multi-modality action* (MMA). An MMA represents in a compact form alternative ways, or alternative *execution modalities*, to accomplish a task. In planning terms, we would say that all the execution modalities of a given MMA reach the same set of propositional effects. However, since they are characterized by specific resource consumption profiles, they differ in their numeric preconditions and effects.

From the MMA formulation we obtain theoretical and practical computa-

[1]Since version 2.1, PDDL allows the employment of numeric fluents in the definition of action templates.

tional benefits; in fact, while replanning is in some case unavoidable, it can be an excessive reaction in many situations as *just* reasoning over the action modalities can be sufficient to overcome exceptions. Moreover, the reconfiguration of the MMAs does not affect the causal structure of the plan. For this reason, if the unexpected deviations can be handled with an alternative configuration of action modalities, the plan remains quite stable. This is important when the agent is situated in a multi-agent setting as humans and/or artificial agents have provided mandatory expectations on the agent's task.

Relying on the notion of MMAs, the paper proposes a system, called FLEX-RR (*FLexible EXecution via Reconfiguration and Replanning*), which is an extension of the on-line approaches based on replanning. The approach is inspired to the continual planning paradigm [2, 7] since it allows the agent to interleave (re)planning and execution. FLEX-RR aims at limiting the replanner's intervention by singling out those situations which could efficiently be resolved via (a simpler) reconfiguration phase. On the other hand, when plan reconfiguration fails or is not sufficient, FLEX-RR starts a replanning task.

Since FLEX-RR encompasses both the reconfiguration task introduced in this paper and a pure replanning approach, it can deal with those situations solvable via replanning but not via reconfiguration. Thus, in the worst cases FLEX-RR is comparable to replanning from scratch. In practice, the reconfiguration can be sufficient to overcome resource consumption anomalies in several cases, as we will discuss in the experimental section.

Section 2 introduces the main concepts at the basis of the FLEX-RR strategy: the MMA model and the Dynamic Modality-Assignment Problem (DMAP), which formalizes the reconfiguration problem. Section 3 presents the implementation of the FLEX-RR strategy. An important aspect addressed in this section is the transformation of a DMAP into a Constraint Satisfaction Problem (CSP), and its resolution via a module called ReCon (ReConfigurator). An extensive sets of experiments are reported in sections 5 and 6; firstly we compare FLEX-RR with two alternative repair strategies: replanning from scratch and LPG-ADAPT ([13]). Then, we analyze the behavior of FLEX-RR and we test the scalability of the CSP solver in handling increasing DMAPs. Finally, we discuss the related works in Section 8.

2 The Dynamic Modality-Assignment Problem (DMAP)

This section formalizes the problem of reconfiguring a plan as a Dynamic Modality-Assignment Problem (DMAP). Before presenting the DMAP and its properties, however, we need to review some basic notions of planning from the point of view of the Multi-Modality Action (MMA) model we propose.

2.1 Modeling the world state

Inspired by the PDDL formalism [10], we abstract the world as a finite set U of physical objects present in the environment, and as properties and relations among such objects. These properties and relations can be both qualitative and quantitative, more formally:

Definition 1 (World Domain)**.** *The world domain is a triple* $< U, F, X >$*, where* U *is the set of physical objects,* F *is the set of qualitative (i.e., propositional fluents) relations involving objects in* U*, and* X *is the set of quantitative (i.e., numerical fluents* [2]*) relations over* U*.*

F and X define the domain in which qualitative and quantitative properties can be stated. This means that the domains of F and X consist of all the possible instantiations of relations w.r.t. the objects in U.

Given the domain defined above, a system state specifies the qualitative and the quantitative properties of the objects in U in that state of the system, formally:

Definition 2 (World State)**.** *A state* S *is a pair*
$< propFluents, numFluents >$*, where:*
$propFluents \subseteq F$ *is the subset of propositional fluents that are true in this state, and*
$numFluents$ *is an assignment of real values to all the numeric fluents in* X*.*

The fluents in F not mentioned in $propFluents$ for a given state S are considered false (Closed World Assumption) in S.

Note, moreover, that $numFluents$ can be also seen as a total function which maps each numeric fluent to a specific real value; for this reason, $|X| = |numFluents|$.

To simplify the notation, in the following we will denote the propositional and the numerical part of a state S by S_{prop} and S_{num}, respectively. Note that, while the propositional fluents in S_{prop} give information about the relations existing between domain objects (e.g. `(ON A B)`), the numeric fluents in S_{num} refine the description of the objects with further properties. In particular, in our discussion we exploit the notion of numeric fluent to model the amounts of objects' resources (e.g. `(= (power r1) 100)`).

In the rest of the paper, we will exemplify our approach by means of the *ZenoTravel* domain: a logistic scenario used in the competitions for testing the ability of planners in dealing with numerical fluents [3]. The *ZenoTravel* domain involves a set of planes in charge of transporting people among a given set of different locations; action models make use of both propositional and numeric fluents. For example, given one plane, two persons and two locations (i.e., $U =$ {*p -plane, p1 p2 -person, a1 a2 -location*}), a system state S as in Definition 2 for the *ZenoTravel* domain is:

[2] Numerical fluents are n-ary function symbols mapping n objects of a domain (or variables when used in action schema) to a real number

[3] http://planning.cis.strath.ac.uk/competition/

S_{prop}: {*(in p a1) (in p1 a1) (in p2 a2)*}
S_{num}: {*(= (fuel p) 100), (= (tot_fuel_used) 0),*
(= (capacity p) 250)}

2.2 The Multi-Modality Action Model

Taking into account the notion of system state introduced above, we now discuss how the system state evolves as an effect of the action application; in particular, this subsection introduces the Multi-Modality Action model (MMA).

As anticipated in the introduction, execution modalities are intended to point out the different ways in which the *same* action can be performed, e.g., using different devices and/or parameter configurations. The intuition is that the expected results of an action can actually be obtained in many different ways by setting *how* the action is configured. Obviously, different configurations have, in general, different resources and time profiles. Therefore, it may be possible that the execution of an action in a given configuration leads to a plan failure, whereas the same action performed in another configuration does not.

The MMA model extends the PDDL action model by allowing to express, within a single action schema, the different alternative modalities in which that action can be performed.

More precisely, an MMA splits an action model into a qualitative and a quantitative description. The qualitative description involves the propositional behavior of an action model and is independent of the modalities. Whereas the quantitative description involves the numeric fluents and is made dependent on the modalities associated with the action. The idea is to model how the selection of a specific modality causes variations (in numeric/metric terms) in the resource profiles, especially for consumable resources[4]. Formally, we define an MMA as follows.

Definition 3 (Multi-Modality Action)**.** *Given the world domain* $< U, F, X >$*, a Multi-Modality Action (MMA) a is the tuple* $< propPre, propEff, mods >$ *where:*

- $propPre$ *is a conjunction of propositions defined over* F*; it describes the applicability conditions, in propositional terms, for* a*.*
- $propEff$ *is a conjunction of propositions defined over* F*, expressing the effect of the application of* a*.*
- *mods is a collection of modalities. Each modality* m *defines a specific way of performing* a *and is modeled as a pair* $< numPre, numEff >$ *where:*
  - $numPre$*: is a conjunction of comparisons that must be satisfied for applying MMA* a *in a state* S *with modality* m*. Each comparison in* $numPre$ *is a triple* $< exp1, comp, exp2 >$ *where* $exp1$ *and* $exp2$

[4]To avoid confusion on the use of the term, with consumable we intend resources which can be produced and consumed

are numeric expressions defined over real constants and over X, and $comp \in \{<, \leq, =, \geq, >\}$

- $numEff$*: is a conjunction of assigners which specifies how numeric variables change by applying MMA* a *with modality* m*. Each assigner is a triple* $< op, f, exp >$ *where* $op \in \{increase, decrease, assign\}$*,* $f \in X$ *and* exp *is a numeric expression involving real numbers and fluents in* X*.*

The numeric expressions we deal with are recursively defined as:

- a real number in $\Re$ is an expression;
- a numeric fluent in X is an expression;
- given two expressions $exp1$ and $exp2$, then also $exp1$ op $exp2$ is an expression, where op belongs to {+,*,/,-}.

As many other representation formalisms handling numeric expressions (see for instance [17] and [12]), also in our framework we limit ourselves to linear expression, for two main reasons: (i) to comply w.r.t. state of the art numeric planners, and (ii) for guaranteeing efficiency of the reconfiguration task, which makes use of a CSP encoding (see Section 2.4 and Section 3).

Given the model defined in 3, an MMA to be executed in a state has to take into account both of its propositional preconditions and the numeric constraints associated with the modalities of execution. More formally:

Definition 4. *[MMA Applicability] Given a state* S*, an MMA* a *is said to be* applicable *in* S *iff:*

1. $S_{prop} \vdash a.propPre$*:* $a.propPre$ *is supported by* S_{prop}*, and*
2. *there exists a modality* $m \in a.mods$ *such that* $a(m).numPre$*, the numeric conditions associated with modality* m*, are satisfied in* S_{num}*.*

Note that, in general, an MMA a is applicable to a given a state S with different modalities. In the following we will denote as $emods(a, S)$ the subset of modalities in $a.mods$ that are enabled in S; namely, $m \in emods(a, S)$ iff $m \in a.mods$ and $a(m).numPre$ is satisfied in S. However, when a is actually applied to S, just one selected modality $m \in emods(a, S)$ is used to predict the successor state S'.

Definition 5. *[Successor State] Given a state* S *and an MMA* a *applicable in* S*, the application of* a *to* S *with modality* $m \in emods(a, S)$ *yields a successor state* S' *as follows*

1. S'_{prop}*:* $S \cup \{propEff^{+}(a) \setminus propEff^{-}(a)\}$
2. S'_{num}*: Each numeric fluent of* S *is copied in S'; for each assigner in* $numEff(m)$*, the numeric fluent* f *appearing as second term is modified according to op and exp. Note that exp is evaluated with reference to the state* S*, and each numeric fluent involved in exp have a specific value in* S*.*

The transition is deterministic and the order in which $numEff$ is considered does not matter, since the evaluation of fluents in *exp* depends just on the state in which the action is applied.

An important characteristic of the MMA definition given above is the neat separation between the propositional and the numeric part. This separation allows us to introduce the notion of *propositional view* of an MMA. More precisely, given an MMA a, the propositional view of a, denoted as a_{prop}, just refers to the propositional segment of the MMA model. The propositional view is essential when one wants to figure out *what* propositional atoms are reached by applying a given MMA without considering *how* (i.e., without taking into account the MMA modalities). For this reason, Definition 4 can be reconsidered under the propositional view of MMAs. In fact, given an MMA a, it is possible to say that a is just *propositionally applicable* to a state S, when its propositional view a_{prop} is applicable to S_{prop}, that is, when just condition 1 is taken into account.

Also the notion of successor state can be restated in terms of the propositional view of MMAs. Given an MMA a and a state S, the *propositional successor state* S'_{prop} is yielded by applying the propositional view of a to the propositional portion of state S, namely S_{prop}. In other words, the propositional successor state is obtained by restricting Definition 5 just to condition 1.

As we will see in the next section, the possibility of reasoning in such two levels of abstraction, allows us to characterize the repair problem as a Dynamic Modality Assignment Problem.

2.2.1 An example of Multi-Modality Action

From our point of view, the *ZenoTravel* domain is quite interesting since in the original version of the model there are two actions (*fly* and *zoom*) that have exactly the same propositional effects, but different numeric effects. Indeed, the two actions have different impacts on the use of resources (in particular on the numeric fluents *fuel*, *total_fuel_used* and *time-spent*[5]).

These two actions have different preconditions, but it is worth noting that the propositional preconditions are exactly the same (the airplane must be in the source airport) and the only differences in preconditions are the ones related to numeric fluents. These two actions can be perfectly captured in our approach with a single action fly with two modalities *cruise* and *zoom*. In particular, the multi-modality action schema of fly, in a PDDL-like language, is shown in Figure 1.

It is easy to see that the propositional preconditions and effects are shared among all the modalities, while the preconditions and effects concerning numeric fluents are specific for each modality.

To make the domain more challenging in the context of MMA, we added to

[5]Note that, as we will see in the next section, since we deal with *sequential* plans of actions, we can handle the time spent by actions as a consumable (not renewable) resource. Therefore, the time can be modeled as a numeric fluent and its meaning has not to be confused with the notion of time-stamp in the context of temporal planning ([10]), or in planning via timelines ([4])

```
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :modalities {cruise, zoom}
 :precondition (located ?a ?c1)
 :numPrecondition
         (cruise:
            (>= (fuel ?a) (* (distance ?c1 ?c2) (cruise-burn ?a)))
         (zoom:
            (>= (fuel ?a) (* (distance ?c1 ?c2) (zoom-burn ?a))))
 :effect (and (not (located ?a ?c1))
              (located ?a ?c2)))
 :numEffect
         (cruise:
            (and(increase (total-fuel-used) (* (distance ?c1 ?c2) (cruise-burn ?a)))
            (decrease (fuel ?a) (* (distance ?c1 ?c2) (cruise-burn ?a)))
            (increase (time-spent) (/ (distance ?c1 ?c2) (cruise-speed ?a)))))
         (zoom:
            (and(increase (total-fuel-used) (* (distance ?c1 ?c2) (zoom-burn ?a)))
            (decrease (fuel ?a) (* (distance ?c1 ?c2) (zoom-burn ?a)))
            (increase (time-spent) (/ (distance ?c1 ?c2) (zoom-speed ?a)))))
```

Figure 1: Multi Modality Action template for the `fly` action in the *ZenoTravel* domain.

the original *ZenoTravel* domain two modalities for the *board* and *debark* actions, namely, *normal* and *express* which take different amounts of time (shorter for *express*) and different costs (lower for *normal*).

2.3 The Multi-Modality Plan

We are now in the position for introducing the notions of *multi-modality plan* and of *multi-modality planning task*.

Definition 6. *Given the domain* $< U, F, X >$*, a Multi-Modality Planning Problem (MMPP)* Π *is a tuple* $< A, I, G_{prop}, G_{num} >$ *where:*

- A *is the set of possible MMA instances over* $< U, F, X >$*;* [6]
- $I \in 2^F \times \Re^{|X|}$ *is the initial state of the problem;*
- G_{prop} *is the propositional goal; i.e., a conjunction of propositions in* F*;*
- G_{num} *is the numerical goal; i.e., a conjunction of numerical constraints over* X*.*

[6] Since we are interested in plan execution, we consider in this formulation instantiated actions only. However, our implementation supports action schema too. The extension is straightforward.

Note that our notion of MMPP is similar to the Numerical Planning Problem (NPP) proposed by Gerevini et al. [12], the main difference is that we adopt actions with modalities rather than simple actions. It is easy to see, however, that an MMPP can be translated into an NPP; it is sufficient to *flatten* each MMA a in A by generating a set of corresponding simple actions $am_1, .., am_n$, where each am_i $(i : 1..|a.mods|)$ models the behavior of a when the modality $m_i \in a.mods$ is selected. Analogously, by exploiting the relation between a flat action and the more abstract MMA model, we can perform the opposite conversion: given a plan consisting of flat actions, we can build an MMA plan. It is in fact sufficient to create, for each flat action a, a corresponding MMA a' such that the execution modality of a' is set to meet the numeric preconditions and effects of a.

In the following definitions, we use the notation $\pi_{i \to j}$ with $i \leq j$ to denote the plan segment starting at the i-th action a_i of the plan and ending at the j-th action a_j; when the right bound is omitted (e.g. π_i) the length of the plan is assumed, that is π_i is equal to $\pi_{i \to |\pi|}$. Moreover, $S[\pi]$ denotes the state produced by applying actions of the plan π starting from the state S. Note that, since a plan segment is itself a plan, the notation $S[\pi_{i \to j}]$ represents the (intermediate) state S' reached after the execution of the actions $a_i, .., a_j$ from the state S.

Definition 7. *Given a MMPP* $\Pi =< A, I, G_{prop}, G_{num} >$, *the plan* $\pi = a_0(m_0), a_1(m_1), .., a_{n-1}(m_{n-1})$ *is a solution for* Π *iff:*

i. $I[\pi] \vdash G_{prop}$ *and*

ii. $I[\pi]$ *satisfies* G_{num} *and*

iii. $a_0(m_0)$ *is applicable in* I *and each MMA* $a_{i+1}(m_{i+1})$ *is applicable in the state* $I[\pi_{0 \to i}]$ *(with* $i : 0..n-2$*).*

In other words, a sequence π of MMAs (specified with their modalities) represents a solution for Π when the execution of π transforms the initial state I into a final state $I[\pi]$ such that: $(i.)$ the propositional atoms in G_{prop} holds in $I[\pi]$, $(ii.)$ the numerical constraints in G_{num} are satisfied, and $(iii.)$ each MMA a_{i+1} in the plan is applicable in the intermediate state obtained by applying the plan segment $\pi_{0 \to i}$ to the initial state I.

Relying on the notion of propositional applicability of the MMAs, we can also define a propositional solution for a MMPP as follows.

Definition 8. *Given a MMPP* $\Pi =< A, I, G_{prop}, G_{num} >$, *a plan* $\pi = a_0, a_1, .., a_{n-1}$ *is a* propositional solution *for* Π *iff:*

i. $I[\pi] \vdash G_{prop}$ *and*

ii. a_0 *is propositional applicable in* I *and each MMA* a_i *is propositionally applicable in the state* $I[\pi_{0 \to i}]$.

This means that a plan π is a propositional solution for the problem Π if it reaches the propositional goal G_{prop} in the abstracted propositional view.

2.4 The Dynamic Modality-Assignment Problem

As previously anticipated, the execution of a plan π in the real world can be threatened by the occurrence of unexpected events such as variations in the resource consumptions, or assumptions that, made at planning time, become invalid at execution time. It is therefore essential to detect, while the plan is still in progress, any unexpected deviation from the nominal expected behavior.

Given an MMPP Π=$< A, I, G_{prop}, G_{num} >$, and a plan $\pi = \{a_0(m_0), a_1(m_1), .., a_{n-1}(m_{n-1})\}$ solving Π, let S_i be the *observed* system state obtained after the execution of the first i MMAs in π, and let π_i be the plan segment $\{a_i(m_i), ..a_{n-1}(m_{n-1})\}$ still to be performed; we say that:

- π is *valid* at step i iff the plan segment π_i is a solution for the MMPP $< A, S_i, G_{prop}, G_{num} >$. Namely, the final state predicted by applying the plan segment π_i to S_i satisfies both the propositional and numeric terms of the plan goal;
- π is *partially valid* at step i, iff the plan segment π_i is just a propositional solution for the MMPP $< A, S_i, G_{prop}, G_{num} >$. Therefore, the plan segment π_i just guarantees the reachability of the propositional goal G_{prop}, but not of the numeric goal G_{num};
- otherwise, π is *invalid* at step i: the plan segment π_i cannot be safely applied to state S_i since, for at least one MMA $a \in \pi_i$, the propositional preconditions of a will not be satisfied, and this would lead the system to an undefined state.

During the execution of the plan, it is necessary to determine the status (i.e., either *valid*, *partially valid*, or *invalid*) just before starting the execution of each MMA involved. For this purpose, we do not consider just the effects of the last performed MMA, but we also verify whether the propositional goals G_{prop} and the numeric ones G_{num} can be satisfied in the final state reached by π. We do this starting from the observed state S_i of the world in order to capture both the changes caused by the agent, and the changes caused by the possible occurrence of exogenous events.

The assessment of the actual environment state after the execution of an action is, in general, a complex problem since the environment is, in most cases, not fully accessible, and only partial observations about the world are available. The problem of plan execution in partially observable environments has been dealt with in [27] [36]. In this paper, however, we assume that the observability level is sufficient for measuring the effects of an action $a(m)$ (i.e., applied with modality m), and for evaluating the applicability of the next action (i.e., whether the propositional and numerical preconditions of the next action are satisfied).

In principle, whenever the plan π is no longer *valid*, a replanning mechanism should be invoked in order to find an alternative way to reach the desired goals. As we have seen, however, the clear distinction between the propositional and numerical aspects within an MMA allows us to distinguish between a completely invalid plan (which is no longer executable because some of the

required propositional fluents are missing), and a *partially valid* plan which is just propositionally consistent but not numerically.

The idea is that, while an invalid plan can only be repaired by means of a (possibly expensive) replanning step, a partially valid plan can be repaired more effectively by reconfiguring its MMAs. That is, we try to reuse the effort made for the synthesis of π by adjusting the way in which the MMAs in π will be performed. We therefore propose to repair a *partially valid* plan not via a replanning step but via a reconfiguration phase, and to do this, we now formally introduce the notion of Dynamic Modality-Assignment Problem (DMAP).

Definition 9 (The Dynamic Modality-Assignment Problem (DMAP))**.** *Let* $\pi = a_0(m_0), .., a_i(m_i), ..,$
$a_{n-1}(m_{n-1})$ *be a solution for the MMPP* $\Pi =< A, I, G_{prop}, G_{num} >$*, and let* S_i *be the last observed state of the environment (i.e., we are at the* i*-th execution step).*

The Dynamic Modality Assignment Problem Φ *is a tuple*
$< \pi, i, S_i, G_{num} >$ *such that:*

- π *is a plan whose first* i *MMAs have been already performed,*
- π *is partially valid at step* i*,*
- S_i *is the observed status of the environment after the execution of MMA* a_{i-1}*,*
- G_{num} *is the set of numeric constraints to be fulfilled.*

Definition 10 (Solution of a DMAP)**.** *The solution (if any) of a DMAP is a new plan*
$\pi' = a_0(m'_0), .., a_i(m'_i), .., a_{n-1}(m'_{n-1})$ *such that:*

- *for each MMA* a_j*,* $0 \leq j < i$*,* $m'_j = m_j$*: the reconfiguration cannot change the modality of an action that has already been performed;*
- *for at least one MMA* a_j*,* $i \leq j < n$*,* $m'_j \in a_j.mods$ *differs from* m_j*;*
- $S_i[\pi'_i] \vdash G_{num}$*: the new assignment of modalities satisfies the numerical portion of the goal.*

It is worth noting that the solution π' of a DMAP has the same sequence of actions as π. The two plans, π and π', just differ each other in the modalities associated with the actions not yet performed. Note also that, since the input plan is partially valid, the solution of a DMAP will differ from the original setting of modalities at least for one of them.

Observing the DMAP definition, it is quite clear that the search space generated by the problem is exponential in the number of the MMAs still to be executed. Given $card(j)$ the number of modalities associated with the j-th action and i the current step, the search space is hence $\prod_{j>=i}^{j<|\pi|} card(j)$, which is of course bounded by $M^{(|\pi|-i)}$, where M is the cardinality of the larger set of

execution modalities through the DMAPs. As we will see in the experimental phase, even if exponential, the search space can be efficiently handled by the CSP solver employed in our system.

The problem is 'dynamic' as it can arise at any step of the plan execution, and assignments made at given step could be reconsidered at a following one.

Having solved the DMAP we are sure that the new MMA plan is a valid solution for the overall MMPP; that is:

Theorem 1. *Let the plan π be* partially valid *at step i wrt the MMPP Π. If π' is a solution for the DMAP $\Phi =< \pi, i, S_i, G_{num} >$ then π' is valid at step i.*

Proof. To show that π' is valid we have to prove that $S_i[\pi'_i]$ supports both G_{num} and G_{prop}. The former comes directly from the definition of a solution for the DMAP, whereas the latter follows by the fact that π' inherits the causal structure of π (i.e., the qualitative/propositional part of the plan does not change when computing the solution). For this reason π' is computed from a *partially valid* solution and hence $S_i[\pi'_i]$ supports G_{prop}. □

Theorem 1 assures us that solving a DMAP does restore the nominal plan execution. The new plan π', solution for a DMAP instance, is in fact a repaired version of the original plan π.

It is easy to see that the solutions space of a DMAP is a subset of the solutions space defined by MMPP. It is therefore easy to see that a solution for the DMAP, if any, is also a solution for the MMPP. However, the opposite does not hold, namely when the DMAP has no solution, the MMPP might instead have a solution.

The following theorem shows that, when the DMAP fails, the solutions to the new MMPP must differ from the original plan for at least one action.

Theorem 2. *Let π_i be a partially valid plan at step i, and let us assume that the DMAP $\Phi =< \pi, i, S_i, G_{num} >$ has no solution. Let π' be a solution for the MMPP $\Pi' =< A, S_i, G_{prop}, G_{num} >$. Then π' satisfies at least one of the following statements:*

- *there is at least an MMA in π' which does not belong to π_i.*
- *there is at least an MMA in π_i which does not belong to π'.*
- *the order of MMAs in π' differs from π_i, i.e. there are at least two actions a and b where both $a,b \in \pi'$ and $a,b \in \pi_i$ such that if $a \prec b$ in π' then $b \prec a$ in π_i.*

Proof. The proof proceeds by contradiction. Let us assume that the solution found for Π' is such that none of the three statements hold. If it is the case the solution π' will have exactly the same structure of π_i in terms of MMAs contained as it has no different action (first and second statements) and the order is the same (third statement). It is easy to see hence that the only way in which π' differs from π_i is in the modality assigned to each action.

Moreover, by definition of Π' we know that if π' is its solution, then $S_i[\pi']$ supports both G_{prop} and G_{num}. Of course, if π' has the same structure of π_i it means that there will be an assignment of modality in π_i such that $S_i[\pi_i]$ supports G_{num}. On the other hand, by hypothesis, we know that the DMAP does not found any assignment of actions modality for the piece of plan π_i which obviously is in contradiction with the fact that there is an assignment in π_i such that $S_i[\pi_i]$ supports G_{num}. □

While theorem 1 states that in some situations a MMPP can be solved as a DMAP, here Theorem 2 states that when a MMPP cannot be solved as a DMAP, then a solution π' for the MMPP at hand can be found only by changing the structure of the original plan π; namely, by adding/removing MMAs by means of a planning phase. This supports our intuition that, in order to keep the plan as stable as possible, it is helpful to try resolving a DMAP first, and only when this has no solution try replanning from scratch.

Solving a DMAP: Computational Complexity. Another reason to anticipate the MMPP with the DMAP is that solving a DMAP is computationally easier than solving a planning problem. To understand why, let us remember that the classical planning fragment (i.e. STRIPS) is PSPACE-complete (see [3]). Despite there is no complexity result (to the best of our knowledge) on the numeric extension of the classical paradigm, the numeric planning is at least as complex as a STRIPS. MMPP is representative of the numeric planning. In other words, the complexity class of the decision problem *NUM-PLAN-EX*[7], referring to verify whether a given MMPP is satisfiable, includes PSPACE.

On the other hand, the DMAP characterization given in this section is less complex. In fact it is easy to show that:

Theorem 3. *Let ASSIGN-EX be the decision problem of determining whether an instance of a DMAP is satisfiable, ASSIGN-EX is in NP.*

The proof of the NP membership comes from the polynomial conversion mechanism discussed in Appendix A, where we show that a given DMAP can be transformed into a CSP, which is an NP-complete problem. That is to say, *ASSIGN-EX* ∈ NP.

Given the proposition above we can also state that:

Corollary 4. *If the hypothesis that PSPACE ⊃ NP holds, then Class(NUM-PLAN-EX) ⊃ Class(ASSIGN-EX).*

The important consequence of Corollary 4 is that solving a DMAP is simpler, from a computational complexity point of view, than solving an MMPP. This gives us a formal guarantee that, in the worst case, reconfiguring the modalities of an MMA plan is simpler than replanning from scratch. This theoretical result is also confirmed experimentally, as discussed in sections 5 and 6.

[7]Let us refer to such a class by means of Class(*NUM-PLAN-EX*)

3 FLexible EXecution via Reconfiguration and Replanning (FLEX-RR)

So far, we have pointed out that a plan can be made *partially valid* or *invalid* by unexpected contingencies occurring during the execution phase. We have also suggested that, when the plan becomes *partially valid*, the agent can try to solve a DMAP to reconfigure action modalities. On the other hand, when the plan becomes *invalid*, the agent has to find a completely new course of actions (i.e. replanning from scratch).

In this section, we present the *FLexible EXecution via Reconfiguration and Replanning* (FLEX-RR) methodology, and discuss how it can be implemented by exploiting CSP techniques ([37]).

Before that, we provide some hints on how a DMAP is translated into a CSP (a more detailed discussion is reported in Appendix A).

3.1 CSP Variables and Constraints

In CSP-based classical planning [8, 22], the CSP representation requires two sets of variables: one to model the actions, and the other to model the states generated by the action execution. Analogously, also the encoding of a DMAP into a CSP representation relies on two sets of variables:

i. $MODs$ is the set of modality variables: the solution of a DMAP is in fact an assignment of modalities to the plan actions, thus action variables become modality variables;

i.i. $NUMs$ is a set of state variables just encoding the numeric fluents of the original DMAP problem, while propositional fluents are not considered.

As in CSP-based planning (based on a graphplan-like structure [1]), we duplicate the state variables as many times as there are "levels" in the plan π[8].

To capture only those assignments of MODs which are consistent with the actions and the problem at hand, each state level is logically bound with the preconditions of the MMA belonging to that level l, and with the effects of the MMA at level $l-1$. State variables at level 0 are set to represent the initial state of the world, while variables at the last level (the level $|\pi|$) must be constrained with the numeric part of the plan goal.

To limit the search space of the arising satisfiability problem, we consider as decisional only the MODs variable. As matter of facts, the numeric profile throughout the plan can be determined as the implication of the modality selection.

For the sake of clarity, Figure 2 summarizes the variables and the constraints involved in a plan of length n; in particular the figure describes:

[8] Of course, since we are not solving a planning problem, but a DMAP, we do not need the step of graph expansion as the sequence of actions is already known.

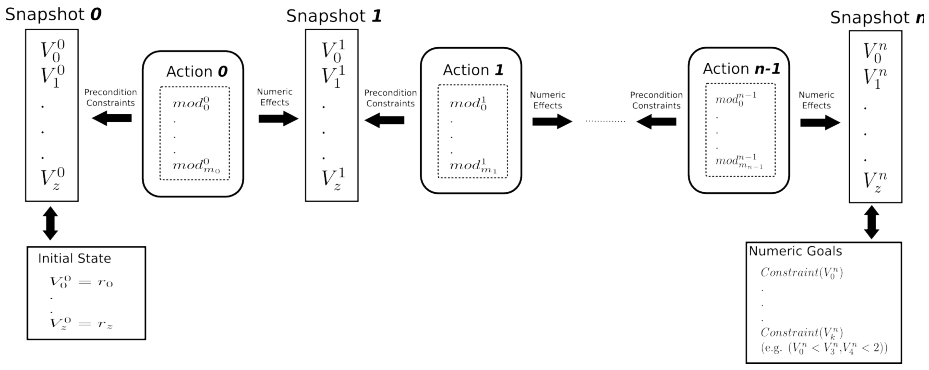

Figure 2: A CSP representation for a generic MMA plan of length n in a problem with z numeric fluents and k goal constraints.

- State Variables $NUMs$ in the *snapshot-layers*. For each numeric fluent N_i in X, a variable V_i^j (with j from 0 to $|\pi|$) is added in $NUMs$.
- Modality Variables $MODs$ in the *action layers*. For each MMA a_k belonging to the plan, a modality variable mod_k, taking values in $mods(a_k)$, is included in $MODs$.

Intuitively, the superscript j of a variable V_i^j in $NUMs$ represents the execution step the variable refers to. Thus, all the numeric variables labeled as 0 represent the initial state; whereas variables labeled as $0 < j \leq |\pi|$ represent (the numeric portion of) the state after the execution of action a_{j-1} with modality mod_{j-1}.

3.2 FLEX-RR: Main loop

Having defined the operative way to handle the DMAP using a CSP encoding, we can now introduce the continual planning approach the agent follows to adaptively perform its plan.

Algorithm 1 sketches the main steps of such a control loop. The basic idea is similar to the continual planning approach [2, 7] in the sense that, at each execution step, the algorithm verifies whether the conditions to achieve the goal are satisfied or not. As innovation w.r.t. traditional continual planning systems, we distinguish between *partially valid* and *invalid* plans, thus we have different strategies for the plan adaptation.

At the beginning, the algorithm initializes two important structures: the $CSPModel$ and the state S. The $CSPModel$ involves all the variables and the constraints for the current plan of actions. Initially it is built by considering the MMPP Π at hand (see Appendix A). As we will see, during the execution of plan π, solving the problem Π, the $CSPModel$ will be modified by adding and deleting constraints, or by asserting new information coming from the system. The state S is initialized according to the initial state I of the problem; also

Input: I, G, π
Output: Success or Failure

```
1  CSPModel=build-CSPModel( I, G, π)
2  i=0
3  S=I
4  while i < |π| do
5      senseAndUpdate(i,S,CSPModel)
6      plan-status = propagate(S, G,π)
7      if plan-status is valid then
8          a_i = getActionAt(π, i)
9          execute(a_i)
10         i++
11     else if plan-status is invalid then
12         π = Replan(S,G)
13         CSPModel = buildCSPModel(S, G, π)
14         i = 0
15     else if plan-status is partially-valid then
16         π = ReCon(CSPModel, i, π)
17         if π = ∅ then
18             π = Replan(S,G)
19             CSPModel = buildCSPModel(S, G, π)
20             i = 0
21 if S ⊢ G then
22     return Success
23 else
24     return Failure
```

Algorithm 1: FLEX-RR

this structure (S) will be updated during the plan execution by the acquisition of information from the environment.

The algorithm iterates over the plan actions as long as there is at least one action to execute. The algorithm returns either *Success* or *Failure* depending on whether the status S reached after the execution of π satisfies or not the goal G. The iteration may be also interrupted when the replan mechanism is not able to find a solution. In that case the replan returns a plan of size 0, so the while condition is not satisfied and a *Failure* is returned.

At each iteration, the algorithm observes and updates the world state S (the *senseAndUpdate* function). Accordingly, also the $CSPModel$ structure is updated at this step with new observed information; in this way, the CSP representation includes all the relevant pieces of information for solving a new DMAP whenever it arises. After the observations gathering, the algorithm assesses the state of the plan π, i.e. it evaluates whether the plan is still *valid* or not. To

Input: i, S, $CSPModel$
Output: updated S and $CSPModel$
Obs = <SenseWorld>
$S = updateStatus(S, Obs)$
if $i = 0$ **then**
  addConstraint($CSPModel, V^0 = S$)
else
  addConstraint($CSPModel$, mod_i=$exec$)
  addConstraint($CSPModel, (mod_i = exec) \rightarrow (V^{i+1} = S)$)

Algorithm 2: SenseAndUpdate

accomplish this step, function *propagate* is invoked to estimate the impact of the updated state S into the planning problem Π. Intuitively, the propagation verifies whether the goal G can be achieved from state S by performing all the remaining actions in π_i without any change in their modalities. In the positive case, the plan is *valid* and its next action a_i is selected for the execution with the modality it is currently associated with. Otherwise, the plan is either *invalid* or *partially-valid*[9].

In case the plan is *invalid*, a replanner is invoked to build a new plan from the current state S to the goal state G[10]. Note that, when a new plan π is returned, this plan substitutes the old one, so the execution restarts from the first action of the new π. For this reason, both the counter i and the model $CSPModel$ need to be re-initialized.

In case the plan is *partially valid*, we have detected a DMAP problem, and FLEX-RR tries to solve it by invoking the reconfiguration module (ReCon). ReCon finds (if exists) an alternative assignment of the modalities to the actions still to be performed. Note that, since the DMAP problem might not have solutions, ReCon could return an empty plan; thus, also in this case, the algorithm will invoke a replanner to resolve the impasse.

The next two algorithms explain in detail: (i) how the CSP model is updated along the plan execution, and (ii) how the CSP solver is invoked.

3.3 Updating the state and the CSP model

Algorithm 2, *SenseAndUpdate*, takes in input the index i of the last performed action, the current state S of the world, and the CSP model to update.

First of all, new observations Obs, collected from the environment, are used to update the state S. Then, the algorithm updates the CSP model depending

[9] Of course, it is important to note that the propagation machinery must take care also of the precondition all along the plan being executed. The mechanism can be improved by exploiting the numeric kernel notion defined in [38]

[10] To allow the interaction between our software module with a generic PDDL numeric planning (Metric-FF, [17] in our case), we apply the conversion mechanisms sketched in Section 2. When the planner is invoked, each MMA model is flattened to the traditional PDDL model; these models are therefore used by the planner. Then, once the plan is computed, the PDDL actions of the plan are newly transformed into MMAs.

Input: π, $CSPModel$, G_{num}
Output: reconfigured π or $\emptyset$
$Solution$ = CSP-solver($CSPModel$)
if $Solution = null$ **then**
  return $\emptyset$
else
  reconfigurePlan($Solution$, π)
  return π

Algorithm 3: ReCon

on the fact that it is the first step (i.e., i equals 0) or a subsequent one. At the first step of execution, the CSP model is updated by imposing that each numeric variable in V^0 assumes the value of the corresponding numeric fluent in S, where V^0 is the subset of numeric variables within the CSP model associated with the level *0*. This allows the approach to deal with an initial world state that is different from the assumed one.

In any other execution step (i.e., $i > 0$), the CSP model is modified by changing the modality of the last performed action, a_i, to *exec*. This special modality has two important roles. First, when an action has modality *exec*, it cannot be considered during the resolution of a given DMAP as its modality cannot be changed any more. Second, when action a_i has modality *exec*, the variables in V^{i+1} are no longer constrained. The constraints are in fact defined according to the modalities specified in the MMA model of a_i, and *exec* is not among them. This allows us to assert within $CSPModel$ any observations coming from the real world even though they are completely unexpected (i.e., not foreseen by any modality associated with a_i). Note that in this way we do not need to re-build the CSP model from scratch, but simply adjust the same model progressively at each execution step. When a new DMAP occurs, the CSP model already encodes all the information required for the DMAP resolution.

3.4 Reconfiguring the plan with ReCon

Algorithm 3 shows the high-level steps of the *ReCon* module. In particular, ReCon has to solve a DMAP problem, and takes in input the plan π to be repaired and the CSP model which, as said above, already encodes all the pieces of information relevant for the solution of the DMAP. A solution consists in an assignment of modalities to the modality-variables that are not set to *exec* (i.e. the modality-variables for the actions not yet performed).

ReCon tries to solve a DMAP by means of a CSP solver. If the CSP-solver finds a solution, this is extracted and used to reconfigure the plan π which is therefore returned to FLEX-RR. In other words, each action a_i still to be performed is assigned the modality selected by the CSP-solver (see function *reconfigurePlan* in Algorithm 3). On the other hand, when the CSP-solver does not find any solution, ReCon returns an empty plan.

3.5 An example of how FLEX-RR intervenes

To exemplify how FLEX-RR is able to monitor the execution of a plan and to repair it in case the plan is no more valid, let us consider a simple planning problem from the *ZenoTravel* domain. Our problem P involves 7 entities, among which: three persons (`P1`,`P2` and `P3`), four airports (`A0`, `A1`, `A2`, `A3`) and one airplane `F1`. The location of the persons and of the airplane in the initial state is represented by means of the following propositional fluents:

(in P1 A1) (in P2 A1) (in F1 A1) (in P3 A2)

whereas the resources (modeled as numeric fluents) have in the initial state the following assignments:

(= (time-spent) 0)
(= (fuel) 8000)
(= (total-fuel-used) 0)

The goal has both a propositional and a numeric part:

propositional: {*(in P2 A2)*, *(in P1 A3)*, *(in P3 A3)*}

numeric: {*time-spent*$<$ 21000, *total-fuel-used* $<$ 10000, *(fuel F1)* $>$ 0 }

Let us suppose that the following plan π has been provided as a solution for the planning problem above.

```
0: board_P1_F1(normal)
1: board_P2_F1(normal)
2: fly_F1_A1_A2(cruise)
3: debark_P2_F1(normal)
4: board_P3_F1(normal)
5: fly_F1_A2_A3(zoom)
6: debark_P1_F1(normal)
7: debark_P3_F1(normal)
```

Before starting the execution, FLEX-RR builds the $CSPModel$ structure for the plan at hand. In this specific case, since the plan involves 8 actions (with indexes 0 thru 7), $CSPModel$ contains 9 levels of (numeric) state variables (from 0 to 8). The state variables in level 0 represent the initial state of the world, while the state variables in level 8 represent the state reached after the execution of the plan. Of course, the numeric goals have to be satisfied by these latest variables.

Let us suppose that the execution of the first two actions in the plan does not raise any problem. However, after the execution of action `fly_F1_A1_A2(cruise)` (index `2`), the flight `F1` has reached airport `A2`, but it has consumed a greater amount of fuel and time than expected. As a consequence, the resulting execution state *S3* is different from the predicted one.

FLEX-RR invokes the propagation mechanism to assess whether the final state will satisfy the goal constraints despite this unexpected situation. Let us assume that the propagation step highlights that the fuel will assume a negative value, which of course is not consistent with our numeric goals. However, the propositional goals will be achieved the same in the final state S8. This means that, after the execution of `fly_F1_A1_A2(exec)`, the plan is *partially valid*. Thus, FLEX-RR tries to repair the plan via ReCon instead of invoking the replanner.

ReCon has to solve a new DMAP by finding a new assignment of modalities to the actions not yet executed. For instance, it finds the following reconfigured plan:

```
0: board_P1_F1(exec)
1: board_P2_F1(exec)
2: fly_F1_A1_A2(exec)
3: debark_P2_F1(normal)
4: board_P3_F1(normal)
5: fly_F1_A2_A3(cruise)
6: debark_P1_F1(express)
7: debark_P3_F1(express)
```

It is easy to see that the modality of `fly_F1_A2_A3` has been changed from zoom to cruise in order to reduce the fuel consumption, so that the constraint on fuel is no more violated (see e.g., the MMA model in Figure 1). However, to compensate the delay caused by such a change, also the modality of the actions `debark_P1_F1` and `debark_P3_F1` have been changed from *normal* to *express*, so that also the constraint on time is satisfied. Of course, the actions marked as *exec* are not addressed by ReCon.

After these changes, the plan is again *valid*, and its execution can resume from the current state S3.

It is worth noting that the new allocation of modalities produces a new plan π' which is very close to the original plan π. In Section 4, we will discuss the importance of keeping a repaired plan as stable as possible (i.e., as close to the original plan as possible), and how the stability can be measured.

In the previous example, FLEX-RR has been able to find a solution by means of ReCon without the need of replanning. This is not always the case. Let us suppose that the execution of `fly_F1_A1_A2` (in *zoom* modality), is affected by a very large deviation in the fuel consumption.

Also in this case the plan is *partially valid*, and therefore ReCon is invoked for solving a DMAP. However, since the deviation on the fuel is significant, it is no possible to restore the validity of the plan via a simple reconfiguration. Thus, ReCon fails and FLEX-RR invokes the replanner. The new planning task consists in finding a plan that, starting from the current state S3, achieves the same set of propositional and numeric goals. In this case the replanner finds a solution:

```
0: board_P1_F1(exec)
```

```
1: board_P2_F1(exec)
2: fly_F1_A1_A2(exec)
--replan--
0: debark_P2_F1(express)
1: board_P3_F1(express)
2: refuel_F1_A2
3: fly_F1_A2_A3(zoom)
4: debark_P1_F1(express)
5: debark_P3_F1(express)
```

Note that the new plan segment has substituted the original plan. In particular, thanks to the introduction of a refuel action `refuel_F1_A2`, the new plan achieves the goals satisfying the constraints on resources. Having the new plan, FLEX-RR starts to execute it. In this case, however, FLEX-RR has to re-initialize its structures by resetting the action index i and by building a new CSP model.

Finally, it is worth noting that also the replanner may fail as not all the anomalous situations encountered during the execution are repairable. More important, in many real-world scenarios, the agent must react to unexpected situations in a short amount of time. That is, FLEX-RR has to find a solution within a given threshold; otherwise, the plan goals must be revised and a new plan must be synthesized.

4 Measuring the Stability of the Plan

In the previous section we have seen that in FLEX-RR both the ReCon and Replanning mechanisms can be invoked for repairing the current plan from a failure. One relevant question concerns how different the new plan is from the old (failed) plan. Obviously, the repair process must have produced a new plan that differs, even just slightly, from the old plan in order to overcome the impasse. However, not all the changes induced by the repair process have the same weight. Intuitively, changing just a modality in one MMA should be considered as a "minor change", while the substitution of an MMA with another one should be consider as more intrusive. When the new plan is substantially different from the one synthesized off-line, the behavior of the agent becomes difficult to predict for an external observer, as for instance a human supervisor. As matter of facts, artificial agents (e.g. robotic systems) or humans may have expectations over the agent's tasks, which may not be explicit in the action models. For this reason, there is an interest to have the repaired plan with the minimum amount of changes: this is particularly true in domains where the agent has to cooperate with others agents (both artificial or human) in order to achieve a global common goal. In fact, a relevant change in the services (and/or in the order) provided by an agent to other agents could cause a major reshaping in the plans also of other agents in order to continue to be able to achieve the common goal.

A first answer to the question is provided by Theorem 2, which guarantees that if Replanning finds a repair plan after ReCon has failed, then the new repair plan cannot have the same causal structure of the old one (i.e. at least an action has to be added or deleted, or the order of the actions has to be changed).

This result is relevant since it provides the basis for stating that the repair plans obtained via ReCon are more *stable*, since they preserve the causal structure of the original plan. However, in FLEX-RR both replanning and Recon are invoked and it would also be useful to evaluate whether the insertion or deletion of a single action in the new plan has a more negative impact on the plan stability than the change of the modalities in many actions (as may happen as the result of ReCon).

To provide a formal basis for such an analysis, we propose a measure of the stability of a repaired plan, based on the notion of *distance* between two plans (i.e., the original and the repaired plans[11]).

4.1 Stability

Our stability measure relies on the *distance* $D(\pi', \pi)$ between the new (repaired) plan π' and the original plan π. Such a measure is inspired to the well-known Levenshtein's string distance [21]. We compute such a distance as the minimum cost for transforming π' into π, where a transformation is a sequence of operations, each of which has a positive cost. The operations we consider are:

- inserting (*add*) and removing (*del*) an action in π', with cost α
- replacing (*remod*) the modality of an action in π', with cost γ
- swap (*swp*) the order of two consecutive actions in π', with cost θ

It is reasonable to assume that $\gamma \ll \alpha < \theta < 2 * \alpha$; that is, changing a modality is less expensive than inserting/removing an action, that in turns is cheaper than swapping two actions. Of course, swapping two consecutive actions is less expensive than adding and removing an action. Thus, the transformation $\tau[\pi', \pi]$, transforming π' into π, involves $adds_\tau$ of insert operations, $dels_\tau$ delete operations, $remods_\tau$ replace modality operations, and $swps_\tau$ swap operations. The cost of τ is therefore computed as:

$$cost(\tau[\pi', \pi]) = \alpha * adds_\tau + \alpha * dels_\tau + \gamma * remods_\tau + \theta * swps_\tau$$

Let T be set of all the possible transformations of π' into π, the *distance* $D[\pi', \pi]$ between the two plans is the cost of the *cheapest* transformation in T:

$$D[\pi', \pi] = \min_{\tau \in T} cost(\tau[\pi', \pi])$$

The computation of $D[\pi', \pi]$ can be done in an efficient way by adopting a dynamic programming procedure, similar to the one proposed by Levenshtein,

[11] Other stability measures have been reported in literature, none of them unfortunately takes into account actions with different modality of execution ([9],[11],[33])

that allows us to find the minimal distance without computing the set T first[12]. Having this notion of distance between plans, we can define the stability of π' w.r.t. π. Intuitively, the stability has to be maximum when no change happens, i.e. π' equals π, and minimum when the two plans are completely different.

$$stability(\pi', \pi) = \frac{cost(\tau_{trv}[\pi', \pi]) - D[\pi', \pi]}{cost(\tau_{trv}[\pi', \pi])}$$

where $cost(\tau_{trv}[\pi', \pi])$ is the reference cost of the trivial transformation, τ_{trv}, which first removes all the actions in π', and then inserts all the actions in π to π'.

The stability measure defined by the above equation ranges from 0 to 1. In particular, when $D[\pi', \pi]$ is close to the cost of τ_{trv}, this suggests that the new plan π' is substantially different from the original one, and hence we compute a low grade of stability (close to zero). On the other hand, when $D[\pi', \pi]$ is significantly lower than the cost of τ_{trv}, the two plans π and π' are very similar, and we compute a high grade of stability (close to one).

4.2 Computing the plan stability: A simple example

Let us consider again our running example, and compute the stability of the repaired plans inferred by FLEX-RR (Figure 3) and by REPLAN (Figure 4), in the two situations described in Section 3.5.

ORIGINAL:	REPAIRED:
debark_P2_F1(normal)	debark_P2_F1(normal)
board_P3_F1(normal)	board_P3_F1(normal)
fly_F1_A2_A3(zoom)	fly_F1_A2_A3(cruise)
debark_P1_F1(normal)	debark_P1_F1(express)
debark_P3_F1(normal)	debark_P3_F1(express)

Figure 3: FLEX-RR

ORIGINAL:	REPAIRED:
debark_P2_F1(normal)	debark_P2_F1(express)
board_P3_F1(normal)	board_P3_F1(express)
fly_F1_A2_A3(zoom)	refuel_F1_A3(default)
debark_P1_F1(normal)	fly_F1_A2_A3(zoom)
debark_P3_F1(normal)	debark_P1_F1(express)
	debark_P3_F1(express)

Figure 4: REPLAN

Given that α (the insertion/deletion cost) equals to 5, γ (modality replacement cost) to 1 while θ (the swap cost) is set to 6, we have that:

[12]The complete procedure is described in [39]

- FLEX-RR: the trivial transformation costs 50 (five deletes and five adds are involved), while the distance between the original and the repaired plans is 3 (3 changes of modalities), and hence the stability grade is 0.94, meaning that the two plans are very close to each other.
- REPLAN: the trivial transformation costs 55 (five deletes and six adds are involved), the distance between the original and the repaired plan is 9 (the add of a refuel action and 4 changes of modalities) and hence the stability grade is 0.83.

This means that the plan repaired via replanning is less stable than the plan repaired via ReCon.

5 Experimental Results

To evaluate the performance of FLEX-RR, and in particular the contribution given by the reconfiguration mechanism, we compared FLEX-RR with two architectural variants: FLEX-REPLAN and FLEX-LPG-ADAPT. FLEX-REPLAN differs from FLEX-RR in that it invokes a replanner not only when the plan π_i is *invalid*, but also when the plan is *partially valid*; substantially FLEX-REPLAN is a pure continual planner that ignores the characterization of the plan given in form of MMAs; instead of considering the space of modalities, FLEX-REPLAN adopts a strategy based on a replanning completely from scratch in each case (i.e. as if it was a new MMPP). Whereas, FLEX-LPG-ADAPT substitutes the replanning from scratch mechanism with the plan-adaptation strategy presented in [9]. It is worth noting that, at best of our knowledge, LPG-ADAPT is the only plan adaptation system which is currently able to deal with numeric fluents during the adaptation task.

The three architectures have been assessed with respect to three parameters:

- *Competence*: the capability of a strategy of finding a repair plan when the original plan becomes *partially valid* during the execution. In particular, we measure the competence as the rate of successes in recovery from the unexpected contingency. In our experiments we allotted the various architectures 240 seconds of CPU time, which is a threshold within which a solution must be provided. In the case of FLEX-RR, we set the time for ReCon module up to $\frac{1}{10}$ of the total computational time budget. One order of magnitude of difference has been considered for limiting the impact of ReCon, hence leaving a reasonable amount of time for a possible replanning, which, as it has been seen in section 2.4, has a larger search space to explore[13]. Therefore, ReCon is allowed to find a solution in 24 seconds, after which FLEX-RR switches to Replan[14]. Note that, since both

[13] Further experiments have been run for different ratios or by allotting different amount of CPU time. These experimental results (see http://www.di.unito.it/~scala/software) show a general trend very similar to the one we will deeply discuss in this section.

[14] When Recon proves that a DMAP has no solution in less than 24 seconds, the time not used by ReCon is given to Replan.

FLEX-REPLAN and FLEX-LPG-ADAPT are allowed to search over all the possible actions combinations (and over all their possible configurations), they are in principle more competent than ReCon. However, since a solution must be found within the threshold of 240 seconds, the actual competence depends on the ability of solving the problem in a timely fashion. In particular, given Theorem 3, we expect that in several cases FLEX-RR can take a great benefit from the ReCon mechanism.

- *Efficiency*: we compare the CPU time of the three architectures. In principle, we will expect that the computational effort will be lower when the cases are solved via reconfiguration, as the search space generated by the DMAP is much more limited.

- *Stability*: we are interested in understanding how the systems impact the structure of the plan to be repaired. In the following experiments we used the same setting of weights reported in 4.2. Relying on Theorem 2, we expect that, when the plan is repaired by ReCon, we will keep the plan quite stable, since the causal structure remains unchanged. When the contingency is solved via Replan, we expect a stability reduction since actions can be added/removed freely and also their order can be changed. On the other hand, since LPG-ADAPT is built with the concept of stability in mind, we expect that FLEX-LPG-ADAPT turns out to be quite competitive in this perspective.

Tests were conducted in three numeric domains, extended to support actions given in form of MMA. That is:

- *ZenoTravel*

- *DriverLog*

- *Planetary Rover*

The first two domains are from the Third International Planning Competition[15]; they have initially been introduced to challenge planners in handling numeric fluents. The third domain has been introduced in our recent works on intelligent supervision of space exploration missions [26, 29], while its complete model definition is reported in [39].

For each domain we generated a set of problems varying the number of objects to be considered. For a given problem we generated a plan whose execution is handled by means of a software simulator. Here, numeric fluents used to model time and consumable resources have been noised in order to reproduce unexpected contingencies. More precisely, the simulation of the deviations on the expected state of the system is obtained by altering the impact of each action executed. In this way, we collected a series of situations where the plan became *partially valid* at least once during its simulated execution.

[15] http://planning.cis.strath.ac.uk/competition/

Each parameter (competence, efficiency and stability) has been measured w.r.t. four degrees of noise. The first degree alters the resource consumption of 25% more than expected, the second of 35%, the third of 50%, and the last one of 75%. We aim at understanding how the noise impacts the performance. Intuitively we expect that, as long as the noise increases, the chance of finding a solution decreases, especially for ReCon. Thus the noise degree could have an impact both on the competence and on the stability[16].

For all the domains tested, cases have been split in two classes of difficulty, each determined by the strictness of the constraints. The idea is to assess the behavior of the system w.r.t. the combination of the difficulty of the problem and the amount of noise injected.

In our experiments we have not taken into account situations in which the plan became *invalid*, since we were interested in evaluating the ability of FLEX-RR in repairing *partially valid* plans, i.e. when the reconfiguration can be exploited.

5.1 Software and Hardware Setup

FLEX-RR has been developed in Java; it receives in input the domain, the problem and (optionally) the plan description written in an extended version of PDDL 2.1 level 2, which incorporates the notion of modalities. FLEX-RR exploits and extends a PDDL Java Library, namely PPMaJaL[17].

As a CSP solver we used Choco 2.1.4[18] which is able to efficiently handle complex constraints on a very large set of variables. As a planner we used Metric-FF [17], which supports the expressiveness of PDDL 2 level 2, by converting the MMAs in PDDL 2.1 actions, as explained in [39].

FLEX-REPLAN and FLEX-LPG-ADAPT inherit the same Java implementation of FLEX-RR, and they exploit, respectively, Metric-FF (in a similar way to FLEX-RR) and the Lpg-Adapt system developed for OAKPlan [13]. Lpg-Adapt is used as a black box and, to activate the adaptation capability given the encountered discrepancy, we invoke the system by setting as input the remaining set of flattened PDDL 2.1 actions to be executed.

Experiments ran on a 2.53GHz Intel(R) Core(TM)2 Duo processor with 4 GB (under the operating system: Ubuntu 10.04).

5.2 Experiments in the DriverLog domain

For the *DriverLog* domain, we collected an amount of 1442 cases, equally subdivided into *easy* and *hard* cases. Our domain definition differs from the one of the planning competition in that we have two modalities for loading packages

[16] However, as studied in [30], one of the main factors impacting the performance of numeric planning is the strictness of the constraints. This means that, an increase of the noise impacts not only ReCon, but also Replan and LPG-Adapt because the numeric problem to be solved becomes more complex (see [17],[12]).

[17] http://www.di.unito.it/~scala/software

[18] Choco is a java library for constraint satisfaction problems (CSP) and constraint programming (CP). Visit http://choco.emn.fr for any further information, or see [31]

(i.e., *normal* and *fast*), and two modalities for driving trucks (again, *normal* and *fast*). Intuitively, an action performed in *normal* mode is cheaper in terms of resource consumption, but slow. Whereas an action performed in *fast* mode is more expensive but faster.

The generated problems vary on the number of packages, locations and trucks to consider. The difficulty of a problem is determined by the involved constraints. More precisely, the problems in the *easy* set have just a constraint on the total time of plan execution, while problems in the *hard* set have a further constraint on the maximum amount of power spent. The length of the produced plans ranges from 14 to 90 MMAs.

Competence. The histograms of Figure 5 reports the competence of the three tested systems. The performances have been measured by considering the percentage of cases solved by a system within the deadline of 240 seconds. The cases are organized according to the degree of injected noise, and to their complexity (i.e., *easy* or *hard*).

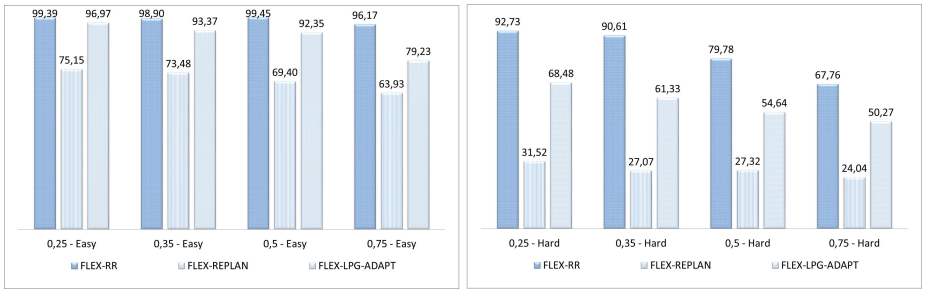

Figure 5: DriverLog Domain: Competence for easy (left) and hard cases (right).

By observing Figure 5, it is quite clear that FLEX-RR outperformed both FLEX-REPLAN and FLEX-LPG-ADAPT in all the given scenarios. In the *easy* set, FLEX-RR was able to repair the plan in almost all the considered cases; a little decrease of the performance (96,17 %) is observable just for the highest degree of noise.

Regarding the *easy* cases, also FLEX-LPG-ADAPT turned out to be quite good. The competence ranges from 92% to 96% in the first three degrees of noise; only when the noise degree is 75%, FLEX-LPG-ADAPT is not comparable with FLEX-RR (79,23% vs 96,17%).

As refers to FLEX-REPLAN, it is evident that its performance is quite negative. In fact, just in the first two degrees of noise, FLEX-REPLAN repairs more than 70% of the cases.

Considering the *hard* cases, the gain between the performance of FLEX-RR and the other two systems becomes more evident; in particular, for noise degree 35%, FLEX-LPG-ADAPT is almost 30% less competent than FLEX-RR, whereas FLEX-REPLAN is more than 60% less competent than FLEX-RR.

FLEX-RR is therefore the most competent in the DriverLog domain. FLEX-LPG-ADAPT performed better than REPLAN, but it is not competitive with

FLEX-RR, in particular for the hard cases set.

Stability. Figure 6 shows the average stability. The set of cases considered by this evaluation refers to those cases which have been solved by all the systems. As reported in Section 4, the stability ranges from 0 to 1, where 1 is the maximum score and the 0 value stands for a repaired plan that is completely different from the original plan. As for the competence, we studied the stability by considering separately the two sets of *easy* and *hard* cases, and the different degrees of noise.

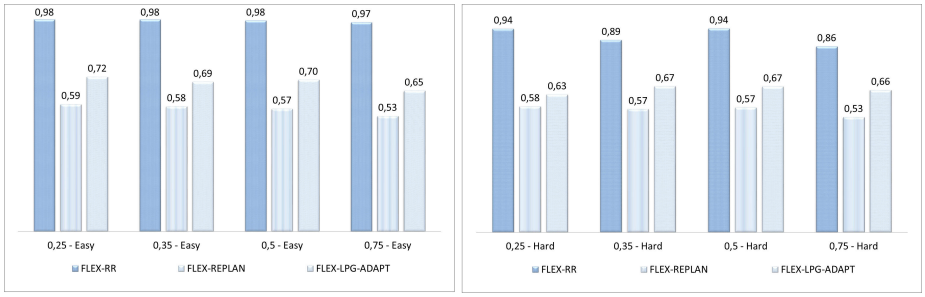

Figure 6: *DriverLog Domain*: Stability for easy (left) and hard cases (right).

Results in Figure 6 show the efficacy of FLEX-RR in keeping the repaired plan stable. FLEX-LPG-ADAPT preserves to some extent the structure of the previous plan, and is more effective than FLEX-REPLAN.

FLEX-RR reaches a stability rate above 0.9 in almost all the scenarios, except that in the hard cases with noise degrees 35% and 75%. As we will discuss in the next section, this result is due to the contribution of the replanning mechanism in the FLEX-RR resolution task.

Efficiency. This evaluation analyzes the CPU time spent by the system in providing an answer, which may be either a positive one (a solution has been found) or a negative one (there is no solution for the problem at hand).

As known in the planning community (see for instance [23]), in most cases the distribution of CPU times cannot be synthesized in few parameters. For this reason, we describe the distribution by reporting the percentage of cases that have been handled in a given interval of time. We have introduced 6 time intervals which partition the interval 0 - 240 sec, plus an additional interval for capturing the timeout of the system. The time has been measured in milliseconds.

The results are in Table 7, which consists of three sub-tables, one for each system. Each row in the table reports the percentage of cases "with answer" in the time intervals above plus the timeout, discriminating between the *easy* and *hard* cases, and among the different degrees of noise. To ease the readability of the table, color shades from grey (low percentage) to orange (high percentage) are used to highlight the intervals where most of the cases are answered.

By observing the results, we can see that FLEX-RR solved the majority of the easy cases in the first interval of time. This means that the system has

		FLEX-RR						FLEX-REPLAN						FLEX-LPG-ADAPT					
	Inf Time Interval	0	101	1001	5001	24001		0	101	1001	5001	24001		0	101	1001	5001	24001	
	Sup Time Interval	100	1000	5000	24000	239999	Timeout	100	1000	5000	24000	239999	Timeout	100	1000	5000	24000	239999	Timeout
	Noise																		
Easy	0,25	98,79	0,61	0,00	0,00	0,00	0,61	3,64	16,97	11,52	20,61	22,42	24,85	3,64	19,39	28,48	20,61	24,85	3,03
Easy	0,35	98,34	0,55	0,00	0,00	0,00	1,10	3,87	10,50	17,13	23,20	18,78	26,52	3,87	16,57	26,52	22,10	24,31	6,63
Easy	0,5	97,81	0,00	1,09	0,00	0,55	0,55	3,28	8,20	12,02	21,31	24,59	30,60	0,55	19,13	24,04	21,31	27,32	7,65
Easy	0,75	93,99	0,00	0,55	1,09	0,55	3,83	4,37	7,65	10,38	19,13	22,40	36,07	2,19	11,48	20,77	21,31	23,50	20,77
Hard	0,25	58,79	24,24	7,88	0,00	1,82	7,27	1,82	4,85	6,67	7,27	12,12	67,27	0,00	14,55	18,79	14,55	20,61	31,52
Hard	0,35	52,49	25,97	8,29	3,31	1,66	8,29	5,52	1,66	6,63	9,39	8,29	68,51	2,21	10,50	17,68	12,71	18,23	38,67
Hard	0,5	45,36	20,22	11,48	1,64	2,19	19,13	6,01	3,28	3,83	7,10	12,02	67,76	1,09	12,57	14,75	10,93	15,30	45,36
Hard	0,75	26,78	21,31	14,21	2,73	3,83	31,15	6,01	3,83	3,83	7,10	8,74	70,49	1,09	10,38	11,48	10,93	16,39	49,73

Figure 7: *DriverLog Domain*: Efficiency for easy (left) and hard cases (right).

provided an answer in less the 100 msec. Whereas FLEX-REPLAN and FLEX-LPG-ADAPT, in most of the cases, required significantly longer. Both systems suffered from the increasing of the noise, and even for the lowest degree of noise (25%), both FLEX-REPLAN and FLEX-LPG-ADAPT needed often more than 5 seconds. By observing the timeout columns we can understand the reason of the low competence of FLEX-REPLAN. In fact, even for the 25% of noise, FLEX-REPLAN did not provided an answer for the 24% of the tested cases.

In the hard setting, the FLEX-RR efficiency remained rather good; most of the tested cases have been solved in less than 1 second for all the noise degrees, making exception for the highest degree where, in the 31,15% of the cases, FLEX-RR reached a timeout.

Proportionally to the increase of the noise, FLEX-LPG-ADAPT became more and more time demanding, ending with 49% of timeout in the 75% setting. FLEX-REPLAN performed even worse reaching the timeout in more than the 65% of the cases in all the four degrees of noise.

5.3 Experiments in the ZenoTravel domain

In our version of the *ZenoTravel* domain we have two modalities of execution for the board and the debark, and two modalities for the fly (as in the original ZenoTravel formulation). Besides the standard behavior, the board and the debark have a second execution modality called "express". The standard modality is cheaper but slower, while the express modality is more expensive but faster. The fly action is already described in Section 2.2.1.

We collected 1036 test cases (518 hard and 518 easy). The generated problems vary from each other on the number of passengers and locations; whereas the difference between the easy cases and the hard ones relies on the number of constraints encoded in the final goal. In particular, the easier cases are constrained in terms of *total-time-spent* while the harder ones add a (further) constraint on the total amount of fuel to be used. Plans involve up to 57 MMAs.

The total number of cases is obtained by running a given problem and plan over the four amounts of noise.

Competence. Figure 8 shows the competence of the systems.

By observing the histogram, for both the *easy* cases and the *hard* ones, FLEX-RR has been more competent than both FLEX-REPLAN and FLEX-

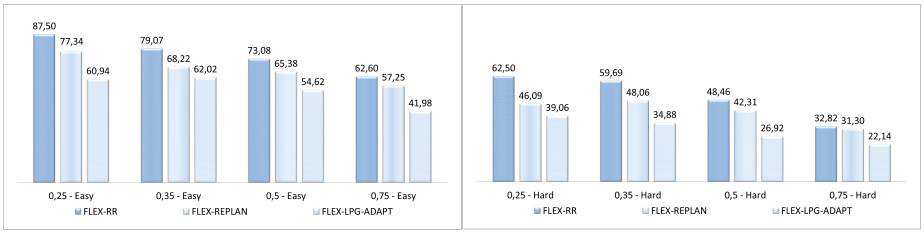

Figure 8: *ZenoTravel* Domain: Competence for easy (left) and hard cases (right).

LPG-ADAPT. In particular, for the *easy* cases, the amount of cases solved with success ranges from 62% to 87% for FLEX-RR, whereas between 57% to 77% for FLEX-REPLAN, and 42% to 61% for FLEX-LPG-ADAPT. For the *hard* cases the advantage of FLEX-RR w.r.t. FLEX-REPLAN increases for the first three degrees of noise; we can observe in fact a gap around the 15%.

As expected, the competence of the tested systems decreases as long as the noise increases.

It is interesting to see that in the last degree of noise FLEX-RR performs as FLEX-REPLAN. As we will see in the next Section, in this domain (in the 75% noise setting, *hard* cases) most of the merit in FLEX-RR is due to the replanning strategy.

Stability. Figure 9 reports the stability measured in the cases solved successfully by all the systems tested. Results proves the ability of FLEX-RR in keeping the structure of the plan quite stable, however the system that produced more stable plans in this setting was FLEX-LPG-ADAPT. Of course, the comparison takes into account only a portion of the cases, because in several cases FLEX-LPG-ADAPT reached the timeout.

Observing the competence and the stability results, it is worth noting that FLEX-LPG-ADAPT is able to solve a case, in particular, for those situations in which the solution is quite close to the starting plan. On the other hand, when the solution is rather distant to the original plan, the performance of FLEX-LPG-ADAPT tends to decrease.

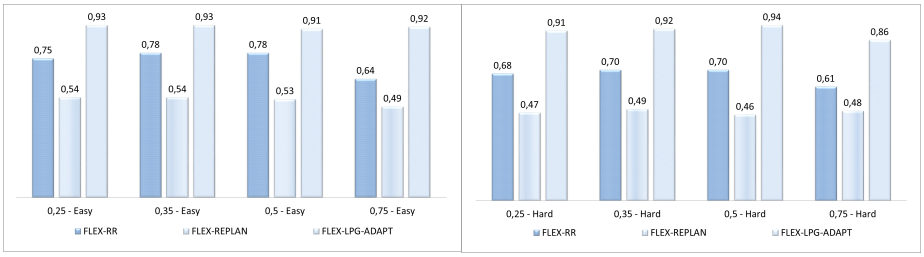

Figure 9: *ZenoTravel* Domain: Stability for easy (left) and hard cases (right).

As expected, FLEX-REPLAN often finds solutions that are quite different

from the original plan.

Generally, the histogram reveals the (quite obvious) relation between stability and noise. That is, as long as the noise increases, the structure of the plan cannot be preserved, and even FLEX-RR produces less stable repaired plans. This is evident especially in the highest degree of noise, where the stability of FLEX-RR is very close to the stability of FLEX-REPLAN.

Efficiency. In Table 10 we can see the cpu time spent by the architectures to handle the partial validity of the plan over all the noises considered.

In each degree of noise, the distribution of cases solved by FLEX-RR (making exception for the first interval of the time with noise degree 25%, where in the 56% of cases have an answer in less than 100s) has been rather uniform. Of course, the population shifts to the right of the table accordingly to the increase of the noise.

Concerning the increase of timeout situations, a similar trend can be found in the behavior of FLEX-REPLAN. In fact the timeouts grow from 16% to 31%. As a difference between the two strategies, we noticed that, when FLEX-REPLAN can solve the task, the resolution process is very fast. Otherwise, FLEX-REPLAN does not provide an answer within the time threshold.

As far as FLEX-LPG-ADAPT is concerned, the low level of competence that we measured before can be explained by the significant number of timeouts reported in Table 10, especially for the *hard* cases.

		FLEX-RR						FLEX-REPLAN						FLEX-LPG-ADAPT					
	Inf Time Interval	0	101	1001	5001	24001		0	101	1001	5001	24001		0	101	1001	5001	24001	
	Sup Time Interval	100	1000	5000	24000	239999	Timeout	100	1000	5000	24000	239999	Timeout	100	1000	5000	24000	239999	Timeout
	Noise																		
Easy	0,25	56,25	18,75	4,69	7,03	7,81	5,47	50,78	14,06	10,94	4,69	3,91	15,62	9,38	33,59	14,84	2,34	0,78	39,06
	0,35	43,41	16,28	7,75	8,53	11,63	12,4	47,29	13,95	7,75	5,43	2,33	23,26	7,75	35,66	12,4	6,2	0	37,98
	0,5	37,69	16,92	6,15	7,69	14,62	16,92	44,62	11,54	10	4,62	4,62	24,62	7,69	25,38	14,62	6,15	0,77	45,38
	0,75	30,53	14,5	6,11	7,63	16,03	25,19	35,88	16,79	6,87	6,11	3,82	30,53	3,05	19,08	12,21	6,11	1,53	58,02
Hard	0,25	30,47	25,78	4,69	3,12	9,38	26,56	32,81	5,47	5,47	6,25	7,03	42,97	7,81	24,22	5,47	1,56	0	60,94
	0,35	24,03	21,71	8,53	7,75	9,3	28,68	35,66	9,30	3,10	6,98	4,65	40,31	3,88	27,13	3,1	0,78	0	65,12
	0,5	16,15	16,92	9,23	5,38	13,85	38,46	33,85	4,62	2,31	9,23	5,38	44,62	2,31	14,62	9,23	0,77	0	73,08
	0,75	10,69	13,74	5,34	3,82	12,98	53,44	25,95	5,34	5,34	4,58	3,82	54,96	0	16,03	3,82	2,29	0	77,86

Figure 10: *ZenoTravel* Domain: Efficiency for easy (left) and hard cases (right).

5.4 Experiments in the PlanetaryRover domain

In our version of the *PlanetaryRover* domain, presented in [25], the most of the MMAs has two or three execution modalities.

In total, we generated 906 test cases: 403 *easy* cases and 403 *hard* cases. Problems vary on the number of locations to be visited and the number of pictures to be taken. The hardness of the problem is controlled by the number of constraints specified in the goal set. In particular the easy case constrains the total time and the power of the rover, while a hard case specifies a further constraints on the communication cost of the mission. In particular, as we will see, this last constraint makes the problem very hard for a generative approach.

The plans involve up to 34 MMAs.

Competence. Figure 11 shows the competence of the three architectures in the *easy* and the *hard* cases. FLEX-RR has outperformed the other systems for all the degrees of noise. In all the noise degrees of the *hard* cases, FLEX-RR has a competence ranging from 84% to 95%; whereas, FLEX-LPG-ADAPT and (still worse) FLEX-REPLAN have very poor performances. In particular FLEX-REPLAN never reaches the 16% of successes (this happens only for the 25% of noise), and FLEX-LPG-ADAPT remains around the 30% of successes; just with the noise degree of 35% the LPG-ADAPT based architecture overcame the 40% of successes.

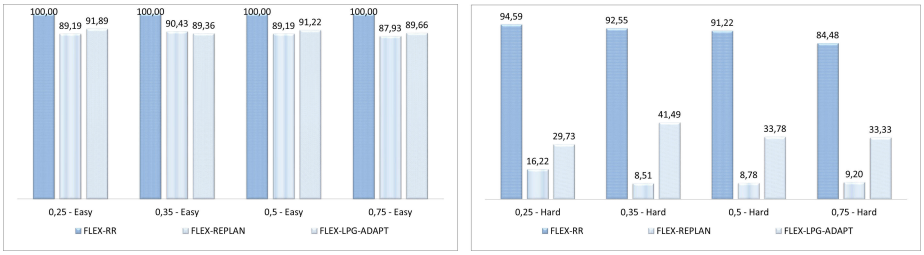

Figure 11: *Planetary Rover* Domain: Competence for easy (left) and hard cases (right).

Stability. The stability of the plans repaired by the systems is reported in Figure 12. Also in this domain FLEX-REPLAN and FLEX-LPG-ADAPT are not comparable with FLEX-RR, which outperforms the other two systems. The plans produced by FLEX-RR have always a stability rate above 0.9 in every noise degree and both for the *easy* and the *hard* cases. Unexpectedly, the plans produced by FLEX-REPLAN are more stable than the plans produced by FLEX-LPG-ADAPT.

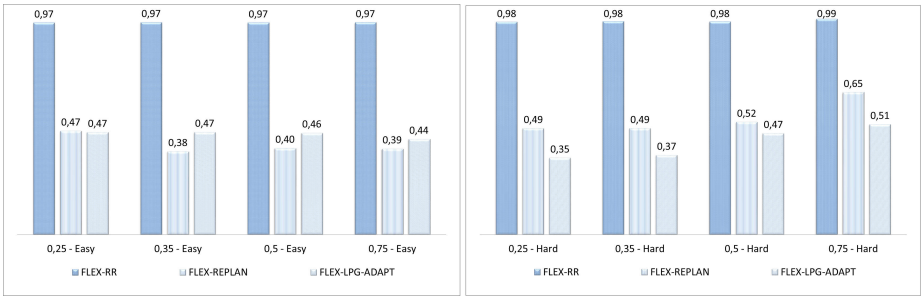

Figure 12: *Planetary Rover* Domain: Stability for easy (left) and hard cases (right).

Efficiency. Similarly to the previous domain, the efficiency results are measured by Table 13. The most of the cases solved by FLEX-RR has been faced in less than 100 msec, for the *easy* cases, and 1 sec for the *hard* cases.

FLEX-REPLAN performed quite well in the *easy* cases, but not so well in the *hard* cases, see the high number of timeouts. Therefore, also in terms of efficiency, neither FLEX-REPLAN or FLEX-LPG-ADAPT has been competitive with FLEX-RR.

		FLEX-RR						FLEX-REPLAN						FLEX-LPG-ADAPT					
	Inf Time Interval	0	101	1001	5001	24001		0	101	1001	5001	24001		0	101	1001	5001	24001	240000
	Sup Time Interval	100	1000	5000	24000	239999	Timeout	100	1000	5000	24000	239999	Timeout	100	1000	5000	24000	239999	Timeout
	Noise																		
Easy	0,25	100,00	0,00	0,00	0,00	0,00	0,00	67,57	8,11	5,41	5,41	2,70	10,81	0,00	21,62	16,22	24,32	29,73	8,11
	0,35	100,00	0,00	0,00	0,00	0,00	0,00	59,57	19,15	5,32	6,38	0,00	9,57	0,00	9,57	12,77	39,36	27,66	10,64
	0,5	100,00	0,00	0,00	0,00	0,00	0,00	59,46	18,92	4,73	4,05	2,03	10,81	0,00	5,41	23,65	33,11	29,05	8,78
	0,75	100,00	0,00	0,00	0,00	0,00	0,00	58,62	18,97	5,17	2,30	2,87	12,07	0,00	4,02	18,97	29,31	37,36	10,34
Hard	0,25	40,54	40,54	10,81	0,00	2,70	5,41	2,70	2,70	0,00	0,00	10,81	83,78	0,00	0,00	0,00	18,92	10,81	70,27
	0,35	45,74	36,17	9,57	1,06	0,00	7,45	1,06	1,06	0,00	4,26	2,13	91,49	0,00	2,13	4,26	10,64	24,47	58,51
	0,5	48,65	30,41	10,81	1,35	0,00	8,78	1,35	0,68	0,68	2,70	3,38	91,22	0,00	0,00	5,41	10,81	17,57	66,22
	0,75	35,63	35,63	10,92	1,72	1,15	14,94	2,30	0,00	0,00	3,45	4,02	90,23	0,00	0,00	4,60	9,77	18,97	66,67

Figure 13: *PlanetaryRover* Domain: Efficiency for easy (left) and hard cases (right).

6 Experimental Results: in-depth analysis of FLEX-RR

While in the previous section we have shown the practical benefits of the FLEX-RR system w.r.t. other configurations, in this section we will focus our attention just on the FLEX-RR architecture. In particular, the next two paragraphs discuss: (i) how the FLEX-RR behaves internally and (ii) how the length of the plan impacts the performance of FLEX-RR, and in particular of its internal module ReCon. The aim is to provide an in-depth analysis of FLEX-RR to single out strengths and possible weaknesses.

6.1 FLEX-RR internal behavior

Analyzing the FLEX-RR strategy reported in algorithm 1, we can see that, while the recovery of an invalid plan can be attempted just via a replanning phase (or via an external plan-adaption tool), the task of repairing a partially valid plan could involve either just a reconfiguration step (when ReCon finds a solution), or the combination of reconfiguration plus replanning (when ReCon fails). As we have seen, this synergy is beneficial in terms of competence, efficiency and stability of the repaired plans.

However, an important question concerns the relative merit of ReCon and Replan in determining the final outcome of FLEX-RR.

In particular the Recon (and Replan) outcome can be:

- Solved: the DMAP (or the MMPP) has been solved and a solution is provided
- No Sol: the DMAP (or the MMPP) was proven to be unsolvable

ReCon	RePlan	FLEX-RR result
Solved	–	Success
No Sol	Solved	Success
Timeout	Solved	Success
No Sol	No Sol	Failure
Timeout	No Sol	Failure
No Sol	Timeout	Failure
Timeout	Timeout	Failure

Table 1: FLEX-RR Success and Failure situations

- Timeout: ReCon (or Replan) did not provide any answer given the time deadline

Therefore we can have 7 possible configurations, each of which corresponds to a Success or a Failure of FLEX-RR. These configurations are summarized in table 1.

By considering all the possible outcomes of Recon and Replan, Figure 14 shows how the cases have been handled respectively for the *easy* and the *hard* cases of the ZenoTravel domain, over all the degree of noise considered (25%, 35%, 50%, 75%). The set of cases analyzed is the same used in the efficiency setting (see Section 5).

As expected, the percentage of cases solved just by ReCon decreases as the noise increases. This percentage ranges from 26% to 57% in the *easy* cases, and from 6% to 39% in the *hard* cases (this last data explains why the FLEX-RR and FLEX-REPLAN have almost the same competence for the 75% noise setting in the hard class, see Section 5). Of course, the case for which ReCon fails (both for No Sol and timeout) are redistributed in the remaining 6 situations. It is worth noting to see that many of them are actually covered by the Replanning system, so the FLEX-RR final outcome is anyway a Success. For instance, in the easy setting, we can see that the percentage of cases for which ReCon proves the unsolvability of the DMAP and the Replan finds a solution increases from 20% to 24%, and, similarly in the hard cases such a range is on the average 17%.

The Figure 14 makes evident that the key for the success of FLEX-RR is the synergy between RecOn and Replanning. Of course such a hybrid behavior comes to a price, which is highlighted by the union of the situations where the ReCon fails and the Replan is successful. It is worth noting that the overhead in CPU time payed by FLEX-RR in such a case is quite modest, since it does not impact neither the overall competence and the efficiency w.r.t. the other configurations (see figure 8 in Section 5). In particular, each case solved by FLEX-REPLAN has been solved by FLEX-RR as well.

Another interesting aspect arising from these results is that, for several cases where FLEX-RR did not provided any solution due to the reaching of the time deadline, ReCon was however able to prove the absence of an alternative re-

configuration of modalities. This is made evident by looking at the percentage of cases relative to the combination "Recon No sol" and "Replan Timeout". While a system equipped just with the Replan capability would have been not informative at all.

Figures 15 and 16 report the result of the in-depth analysis of FLEX-RR in the domain of DriverLog domain and the PlanetaryRover one respectively. Not so many comments are necessary since in these domains most of the merit is of ReCon, in that in just few cases FLEX-RR had to use the replanning mechanism. It can be observed that in domain DriverLog, *hard* cases (Figure 16), there is a relevant number of cases for which ReCon outcome was No Sol and the Replanning ended with a timeout, this shows that ReCon is quicker in detecting unsolvable cases than Replan.

We have also analyzed possible causes why the behavior of FLEX-RR in the ZenoTravel domain is somewhat different with respect to the other two domains. One distinctive feature of the ZenoTravel domain concerns the presence of the refuel action which can renew the consumable resource fuel when the airplane is at a given airport. The presence of a refuel is a very powerful tool for a "generative" approach (as the ones based on a plan adaptation), while it is completely useless for ReCon since it cannot alter the causal structure of the plan. It may be hence the case that, while ReCon spends time in searching for a new configuration of the action modalities, the insertion of a refuel action could be sufficient to repair the plan. This is probably the cause of several situations where ReCon ended with a timeout followed by the success of the Replan. Nevertheless, we also observed that, when the goal involves a constraint on the maximum amount of fuel that can be altogether consumed (e.g., because of the plan has to preserve a bound on the cost of the mission), the refuel action is no longer crucial to repair the plan. This is showed by the decrease of "Recon Timeout"+"RePlan Solved" situations in Figure 14, where the hard cases are considered.

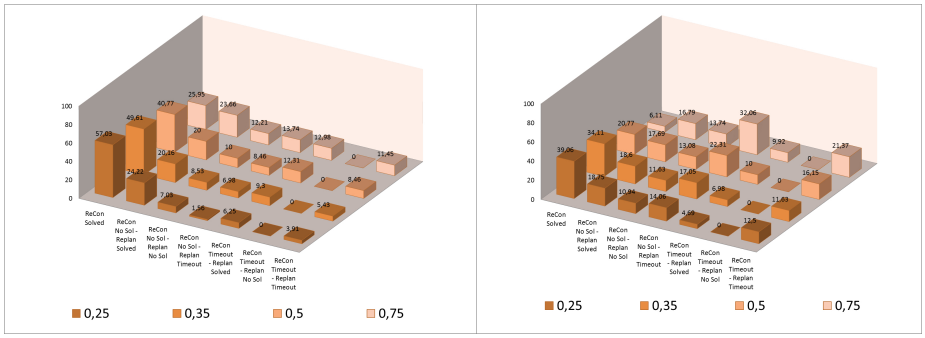

Figure 14: *ZenoTravel* Domain: In-Depth Analysis

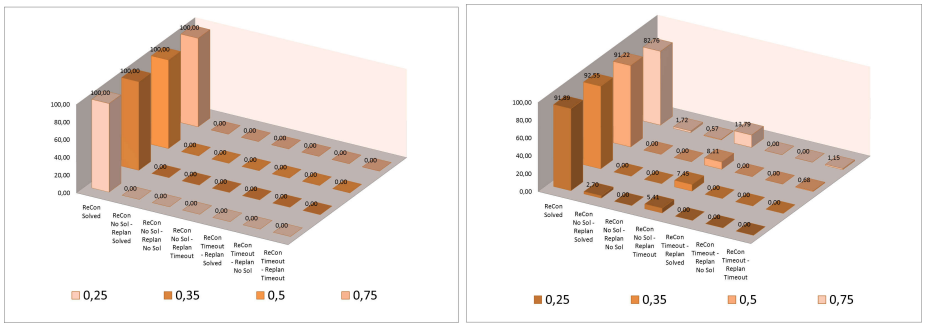

Figure 15: *PlanetaryRover* Domain: In-Depth Analysis

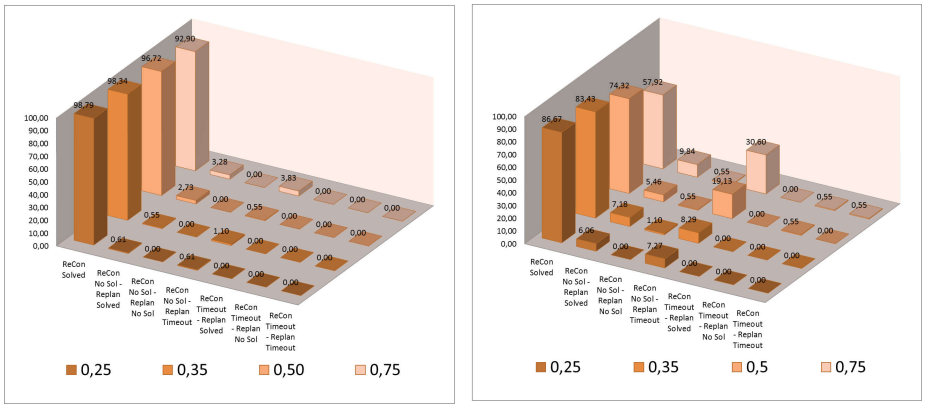

Figure 16: *DriverLog* Domain: In-Depth Analysis

6.2 Scalability

One of the factors which can represent a barrier for the performance of the reconfiguration is obviously the number of the actions in the plan. Actually, in Section 2 we have seen that the search space of a DMAP is exponential in the number of MMA involved in the reconfiguration. Therefore, the actual critical factor is not the length of the plan *per se*, but the length of the DMAP. So it is important to analyze at what extent the CSP is able to deal with (very) large search space when the length of the DMAP increases.

To measure such a parameter, we grouped the test cases of the previous section w.r.t. the number of the actions the DMAP consists of, and with respect to six intervals of time (0-100 msec, 101-1000 msec, 1000-5000 msec, 5000-24000 msec, timeout). Figure 17, 19 and 18 summarizes the results obtained in the three domains under examination. The cases are grouped in subsets of DMAPs of a given length (0-10 actions, 10-20 actions and so forth). This analysis measures hence just the reconfiguration, and the results refers to both the cases solved successfully by ReCon and the cases for which ReCon proved

the unsolvability of the DMAP.

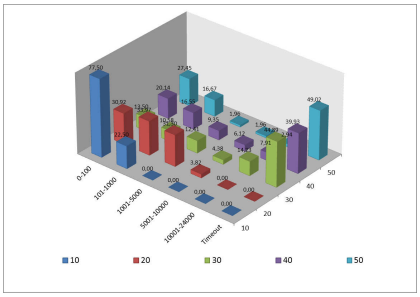

Figure 17: *ZenoTravel* Domain: Scalability

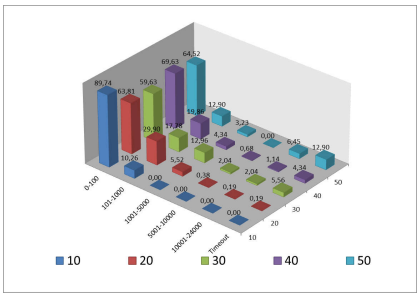

Figure 18: *PlanetaryRover* Domain: Scalability

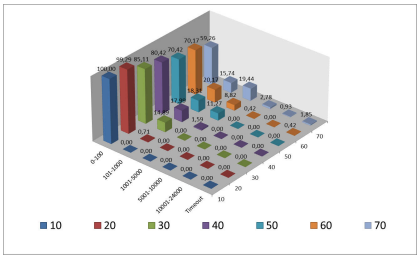

Figure 19: *DriverLog* Domain: Scalability

It is evident that in the DriverLog and PlanetaryRover domains ReCon performs quite well, even for DMAPs involving a large number of MMAs. More precisely, in DriverLog the efficiency of ReCon decreases just for the interval 60-70 actions where the 19% of cases required a CPU time between 1-5 secs. In the PlanetaryRover domain the majority of the cases required no more than 1 sec, and this is true for cases involving up to 40 actions. In the last subset of cases (40-50 actions), the performance remained quite good (in fact the 65% of

cases were solved in less than 100 msec), however a significant number of tests exceeded the time deadline (13%). ReCon behaves differently in the ZenoTravel domain. In fact, for short DMAPs (up to 10 actions) most of the cases are solved in less than 100 msec (i.e., 77% of cases). As expected, the ability of ReCon in solving DMAPs reduces as long as DMAPs get longer. In particular, when DMAPs involve more than 20 actions, the number of timeouts becomes relevant. These results motivate the behavior of FLEX-RR showed in the previous section. In fact it has to take advantage of both ReCon and Replan. Whenever a plan is repairable by means of a reconfiguration of action modalities, FLEX-RR exploits ReCon to efficiently find such a new configuration. On the other hand, when a plan cannot be repaired by changing action modalities, FLEX-RR quickly switches to Replan, which can exploit all the power of a generative planning approach. Note that the quickness in the switch from ReCon to Replan is guaranteed by the fact that either ReCon proves that the DMAP is not solvable, or ReCon consumes all its 24 seconds, which can be considered an acceptable amount of overhead at this level of abstraction for most real-world applications.

7 Extending the MMA model

The MMA model reported so far has been formulated by assuming that (at this planning level) it is reasonable to abstract the behavior of the agent into two distinct levels: the higher describing what the agent does, and the other expressing the "way" in which each action of the plan is executed. This distinction has been reflected in a clear division between the propositional/qualitative schema of the top level and the numeric part referred to the execution modality level.

However, by observing the framework and the CSP computational model reported in Appendix A, it is easy to see that this restriction can be easily relaxed. In some scenarios in fact one would like to constrain the applicability of an execution modality not only to the availability of resources, but also to particular contextual conditions. While some of contextual conditions can be naturally expressed using the same numeric formalism (e.g., the movement of a vehicle is allowed only when the snow does not trespass the 20 cm), others could be much more naturally represented with qualitative constraints, i.e. with propositional constraints (e.g., the vehicle has to be provided with snow chains to follow this road in that execution modality).

To this end it suffices to extend the definition 3. In particular each modality model should be modified with the tuple $< propPre, numPre, numEff >$, where $numPre$ and $numEff$ remains unchanged w.r.t. the previous formulation, while, given a state S, $propPre$ represents a conjunction of propositions that must be satisfied for applying MMA a in a state S with modality m.

The CSP extension at the basis of FLEX-RR is also straightforward; it suffices in fact adding a further constraint (for each modality containing also at least a propositional precondition) all along the (affected) snapshots representing the possible trajectories of states (i.e., the ones prior to the action execution); in addition, the CSP snapshots have to be enriched with the relevant atoms, to

understand whether the modality precondition is satisfied. It is important to remember that the CSP handled online has to intercept all the (relevant) changes in the environment, even if they are uncontrollable or completely unexpected.

It is worth noting that the extension inherits most of the theoretical properties provided so far, so we expect that the competence, the stability and the efficiency will not be compromised.

8 Related Works

The work presented in this paper adopts an action-centered philosophy (in line with PDDL, [10]), differently from a timeline based one ([4],[28]). For this reason, our related works discussion will focus mainly on the relevant literature in this field of research. In particular, three topics are relevant for FLEX-RR: plan repair, CSP-based planning, and plan stability. In the following, we present some reference works in these areas and compare them with our approach.

Plan repair strategies have recently been studied in a number of works (see e.g., [40, 9, 14, 11]).

van der Krogt et al. [40] suggest that plan repair can be seen as a two-step procedure. First, the broken plan is *unrefined* by removing the actions causing plan flaws. Second, the resulting plan is repaired via a refinement step (see [19]) that adds actions to fill the gaps left during the previous phase. The proposed algorithm looks for a solution in the space of possible plans. In case the refinement step does not find a solution, a backtracking occurs and the unrefinement procedure is invoked again.

In [14], Gerevini et al. consider a planning graph structure, and assume that flaws affect different steps of a given plan. The proposed repair strategy solves separately each flaw: first it individuates the relevant plan "window", i.e., the portion of the plan containing the flaw, and then it considers the window as an independent planning task. If the planning task has no solution, the window is enlarged to include one more actions, and the process is repeated. In the worst case, all plan is included within a repair window, and the strategy behaves as a replanning from scratch procedure.

Garrido et al. [11] propose a mechanism to select between repair and replanning. In fact, they heuristically estimate the cost of plan repair (i.e., building just a bridge plan) and replanning from scratch (i.e., building the whole plan to reach the goals), and then solve the least expensive planning task .

The approaches mentioned so far represent a significant step ahead in understanding how plan adaptation can be applied in practice to the plan repair problem, despite contrasting complexity results on the topic ([32]). However, they present limitations when plan repair is to be performed on-line and has to take into account consumable resources and limited amount of time. In fact they are mainly designed for the classical/propositional fragment (making exception for [11] which considers the temporal planning too), and do not provide any theoretical bound on the plan repair task to be faced. Our FLEX-RR methodology overcomes these limitations, both theoretically, by providing a plan repair

formulation given in terms of reconfiguration which is computationally simpler than a replanning approach, and practically by providing an effective architecture which combines competence and efficiency in a unified framework.

Another recent work strictly related to the plan repair problem is [24]. This work, however, is mainly focused on the problem of diagnosing the anomalous execution of a plan. In particular, the diagnosis identifies the root causes for an action failure. These causes are subsequently used to drive a replanning mechanism, based on a conformant planner, aimed at repairing the plan. The main disadvantage of this approach is that the conformant planning phase could be very expensive. The approach could therefore exploit FLEX-RR to repair a plan, at least in same cases, via reconfiguration rather than via replanning.

It must be noticed that, while the approaches mentioned so far do not consider resources, other works ([20, 34, 35, 6]) focus on the temporal dimension. In these works, the notion of robust execution is close to the idea of flexible schedule. The rationale is that, if an agent can arrange its tasks along the time, then the actual execution of the plan is less prone to be affect by unexpected situations. Thus, these works are mainly focused on finding a schedule of the plan actions that guarantees some flexibility to the agent. The problem solved by these works is therefore slightly different from ours. In fact, while they are off-line (i.e., the schedule is synthesized before the plan execution phase), our work is on-line (i.e., a DMAP occurs when the plan execution is in progress). In addition, they are mainly focused on time delays, whereas we deal with exceptions that could threaten any consumable resource, included the time.

The work by Coles [5] is another example of off-line approach in which the planning phase tries to anticipate alternative scenarios that might occur at execution time. The result of this planning phase is a branched plan, where each branch "opportunistically" reaches a rewarded goal. It is interesting to note that new plan branches are created depending on the amount of agent's resource, which are modeled as numeric fluents as in our case. As said above, however, our methodology is on-line and it has not to rely on scenarios anticipated off-line.

FLEX-RR relies on a CSP solver for resolving the DMAPs that possibly arise during the execution. This requires us to transform the planning domain under consideration into an appropriate CSP model. The work done in the planning community for solving classical planning problems via CSPs (see e.g., [22, 8]) has provided us a solid base for developing our translation schema. In the CSP planning encoding, variables represent actions and facts, while constraints are intended to allow only sequences of actions that are valid with respect to the domain and problem specification.

In our case, however, the CSP model is slightly different. Our formulation focuses in fact on the relation between action modalities and resource profiles. Resources are modeled as real variables, which can interact with one another according to the action models (e.g., the power spent by a communication depends on the amount of memory to be transmitted, which in turn depends on the performed sensing actions). It is hence evident that, while our transformation inherits the logical framework of the classical CSP-based planning, a pure propositional conversion is not sufficient to capture our representation.

Finally, in our approach we have also taken into account the notion of plan stability. This important feature has already been considered in previous works. In particular, the work by Fox et al. [9] introduces a measure of the plan stability in terms of common actions. As in our case, the metric is based on the notion of distance between two plans: given two plans π and π' the distance is given by the sum of the number of actions that are in π but not in π' and the number of the actions that are in π' but not in π, over the total number of actions. The same distance is also used in [33] for finding diverse plans. In this paper we have proposed an alternative measure of the plan stability which, besides the common actions, also accounts for the order with which the actions appear and their similarities (where the similarity is captured by the relation between the MMA and the modalities it encompasses). The new metric we propose in fact considers swap of actions as well as replacement of modalities. Meaning that when the order of the actions is significant, our measure is more accurate than the stability proposed in [9], even without the notion of MMA. In [33] there are actually other distance measures upon which the concept of stability can be built. They consider causal relations as well as state trajectories. Causal relations however do not capture change of modalities. As refers the distance based on the state trajectories

It would be interesting to evaluate how the distance based on state trajectories applies to our framework, but it should be refined to deal with states involving numeric fluents, which are crucial in our approach.

Recently, LPG-ADAPT, which is the system developed for [9], has been extended for supporting numeric fluents ([13]). LPG-ADAPT is designed to work in an off-line context, but can be exploited in a continual planning loop as well. For this reason, we integrated the LPG-ADAPT for an alternative configuration of FLEX-RR that we have called FLEX-LPG-ADAPT. Such a configuration substitutes the reconfiguration and the replanning step via the adaptation mechanism presented in LPG-ADAPT. However, as we have seen in the experimental section, in dealing with unexpected resources consumption, this architecture has not been competitive with FLEX-RR due to the lack of knowledge on the MMAs structure. Of course, when the plan to be repaired needs a major revision, FLEX-RR can be easily extended to invoke, at least in some case, the LPG-ADAPT system instead of the replanning from scratch.

9 Conclusions

This paper has addressed the problem of robust plan execution for planning tasks involving (complex) constraints on a number of resources. In coping with this problem, we have considered that the agent (i.e., the plan executor) might require not only reusable resources, but also limited, consumable ones; in our approach, in line with action-centered formalisms, we modeled such reusable and consumable resources as numeric fluents. To the best of our knowledge, previous approaches to robust plan execution that explicitly deal with resources are mainly off-line (e.g., [20, 34, 35, 6, 5]). On-line methods (e.g., [40, 9, 14,

11, 24]), instead, are mostly focused on the achievement of the (propositional) plan goals by interleaving plan execution and (re)planning.

The basic contribution of this paper is the notion of a Multi Modality Action, an extension to the PDDL formalism allowing to encode in the same model of the action the several execution modalities in which such an action can be carried out. Each execution modality models requirements and consequences of executing the action in that execution modality w.r.t. a given set of resources. While all modalities of a given action achieve the same propositional effects, a given modality models "how" these effects can be achieved. According to the Ghallab et al.'s terminology [15], modalities are intended to point out the alternative *operational model* their selection implies. The underlying hierarchical relation defined by the MMA model is an attempt to bridge the gap between the *descriptive* nature of the PDDL language and the operational representation leading the execution of lower level actions [19].

The combination of these two levels of representations allows a new (non-trivial) characterization of the repair problem, formalized as Dynamic Modality Assignment Problem (DMAP). The DMAP is the reconfiguration task of a given plan of actions expressed in terms of MMAs. This formulation has interesting theoretical properties that provides, on the one hand, a guarantee on the computational complexity bound limiting the reconfiguration in the NP class (while a replanning task is at least as difficult as a planning problem, which is PSPACE-Complete, see [32],[3]), and on the other hand, a kind of repair that keeps rather stable the causal structure of the plan. This last property is important especially when the agent has to cooperate with others. High levels of stability, in fact, guarantee that the agent will provide the other agents with the services they are waiting for.

Relying on the MMAs notion and the DMAP formulation, the paper presented the FLEX-RR architecture, which is a continual planner exploiting the synergy of the reconfiguration (i.e. the DMAP) and of a replanning strategy. The objective is to address unexpected variations in the consumption of resources by trying first of all to reconfigure the plan actions, and just in case, when the reconfiguration fails, substituting a plan segment with a new one synthesized on-the-fly.

The advantages of FLEX-RR are easy to see. First of all, when the reconfiguration succeeds, the agent is able to provide a solution in a very efficient way and the structure of the original plan is preserved within the repaired one. On the other hand, when the deviations on the nominal behavior becomes prominent, the agent can exploit the maximum flexibility provided by a generative approach which can add, remove and change the order of the next actions to execute.

FLEX-RR encodes the DMAP as a CSP, and the replanning by means of a numeric PDDL planner. The CSP solver has been used to ensure to find an alternative allocations of modalities that respects the constraints all along

[19] For details on how modalities can be used for controlling the actual execution see also [26].

the plan to be executed, while the numerical planner is aimed at finding a new course of actions given the current observed state and the goal of the planning problem at hand.

To evaluate the benefits of the approach, FLEX-RR has been thoroughly experimented in three planning domains, extension of classical numerical domains from the International Planning Competition. The results confirmed the theoretical advantages provided by the DMAP characterization: in repairing *partial valid* plans, FLEX-RR is more efficient than both a replanning mechanism and the plan-adaptation strategy of LPG-ADAPT [13]; moreover the FLEX-RR repair process keeps the structure of the plan rather stable. More important, given that we used CPU time deadline on the repair resolution task, we measured that FLEX-RR outperformed the other systems even in terms of competence; the gaps on the competence results are very significant in the majority of the (hard) case scenarios.

In future developments of the framework, we will investigate the problem of finding a modality assignment that optimizes a given objective function, such as the maximization of the plan stability or of the resource usage. Also in this case, our hypothesis is that the optimized reconfiguration problem could be easier to solve than replanning optimized w.r.t. a given metric that, as pointed out in [16, 10], is undecidable without any restriction on the planning language.

Another direction of our research is to integrate the feedback of the reconfiguration towards the replanning in a more powerful way. Knowing that ReCon was not able to find a solution in the space of modalities, could in fact be very helpful to control the search strategy of the replanner since some branches of the search space could be discarded.

Acknowledgments

We would like to thank the anonymous reviewers for the comments that helped to improve the quality of the paper, 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.

A CSP Transformation

A.1 CSP Constraints Formulation

Section 3.1 introduced the variables of the CSP formulation for the DMAP problem, and some clue concerning the constraints which has to be built around the variables of our problem. However, the formulation of the preconditions and of the effects in the action model can be complex and may involve many numeric fluents interacting among each other. For this reason, in this appendix we will describe how the constraints binding the variables in $MODs$ and $NUMs$ are built. Let us remember the reader that this step is mandatory for guaranteeing

to obtain only assignments of modalities that are consistent with the model of actions and the problem at hand.

Init and Goal Status Constraints

First of all, let us see how the initial and the goal states of a DMAP can be represented in terms of variables and constraints within a CSP. In fact, the first and the last snapshots of our CSP representation must be constrained in order to be consistent with the initial and the goal states, respectively, of the DMAP formulation. The initial setup of the CSP formulation translate a specific formulation of the DMAP, i.e. $\Pi =< \pi, 0, I_n, G_{num} >$.

1. *Initial State Constraints.* For each *numFluents* in I_n of the form $N_i = val$, the corresponding CSP variable $V_i^0 \in NUMs$ is constrained to assume the value val [20]:

$$\forall\, N_i = val \ \in \ I_n, \ \ CSP.addConstraint(V_i^0 = val)$$

   where val is a constant in $\Re$.

2. *Goal Constraints.* For each comparison of G_{num}, which has the form $C :< exp, comp, exp' >$, where exp and exp' are linear expressions mentioning numeric fluents in X, and $comp$ belongs to $\{<, \leq, =, \geq, >\}$, we create a corresponding comparison mentioning the variables in $NUMs$ at step n, where $n = |\pi|$:

forall the C $\in G_{num}$ **do**
  CSPExp $\leftarrow Subst(C.exp, X, V^n)$
  CSPExp' $\leftarrow Subst(C.exp', X, V^n)$
  $CSP.addConstr(CSPExp, C.comp, CSPExp')$

In the transformation reported above, the procedure *Subst* substitutes the variables present in the goals definition with the variables of the last snapshot of the CSP Model. The result is a CSP expression expressing how the variables of the CSP Model have to be related each other.

Precondition Constraints

As anticipated in section 2.3, an MMA a is said to be applicable in a given state S with modality m only when the numeric preconditions associated with m are satisfied (assuming that the propositional preconditions are satisfied too) in S. To encode the applicability condition within the CSP model, we have to bind each MMA a_i with the CSP variables in V^i; that is, state variables encoding the state in which action a_i has to be executed. Therefore, for each MMA a_i in

for $i = 0 \rightarrow |\pi - 1|$ **do**
  forall the m *in* $a_i.Mods$ **do**
    forall the C $\in numPre(m)$ **do**
      CSPExp := $Subst(C.exp, X, V^i)$
      CSPExp' := $Subst(C.exp', X, V^i)$
      CSP.addConstr($(mod_i = m) \rightarrow$
      $(CSPExp, C.comp, CSPExp')$)

the plan, we add within the CSP model the applicability constraints as follows:

Namely, for each modality m associated with a_i, we consider every comparison C in $numPre(m)$, and translate it into a new expression in terms of CSP variables in V^i similarly to what we have shown for the translation of the goal constraints. The new expression is therefore added to the CSP model.

Effects and Frame Constraints

The execution of an action changes the system state according to its model. In general, however, just a portion of the system state is actually modified by the action; all the numeric fluents that are not directly mentioned in the effects of an MMA are assumed to persist (Frame Axiom).

Therefore, within our CSP model we have to add constraints modeling the transition of each numeric fluent from a snapshot i to a snapshot i+1, taking into account whether the numeric fluent is mentioned within direct effects of the i-th MMA action or not.

- **Affected Numeric Fluents:** for each i-th action in the plan, the selection of the mod_i variable binds the i-th snapshot with the $i+1$ one. More precisely the constraints to be added are:

for $i = 0 \rightarrow |\pi - 1|$ **do**
  forall the E $\in numEff(mod_i)$ **do**
    CSPExp := $Subst(E.exp, X, V^i)$
    CSP.addConstraint($(mod_i = k) \rightarrow (V^{i+1}_{E.nfluent}, E.op, CSPExp)$)

- **Not Affected Numeric Fluents:**

To facilitate the comprehension we will introduce a small example. Let us imagine to be in the position of building the constraints binding the effects of the *fly* action of the *ZenoTravel* in the modality *cruise*.

[20]Note that, an in initial state is a complete state, meaning that there is no numeric fluent with an undefined value

for $i = 0 \rightarrow |\pi - 1|$ **do**
  forall the fluent $\notin affected(mod_i)$ **do**
    CSP.addConstraint($V^i_{fluent} = V^{i+1}_{fluent}$)

The CSP representation will contain 4 numeric variables, i.e. $\{V_{t\_f\_u}, V_{fuel}, V_{t\_s}, V_{capacity}\}$, whereas *distance*, *consumption* and the *speed* are modeled as constant (there is no action affecting them).

By recalling the model of the action reported in Figure 1, we know that the action affects *total-fuel-used (t_f_u)*, *fuel* and *time-spent (t_s)* leaving persistent the others numeric fluents of the domain.

Here the constraints resulting for the i-th fly(cruise) action which is the one that models the plane moving from A to B:

$(mod_{fly} = cruise) \rightarrow (V^{i+1}_{fuel} = V^i_{fuel}$ -
distance(A,B) * avg_cruise_cons)
$(mod_{fly} = cruise) \rightarrow (V^{i+1}_{t\_f\_u} = V^i_{t\_f\_u}$ +
distance(A,B) * avg_cruise_cons)
$(mod_{fly} = cruise) \rightarrow (V^{i+1}_{t\_s} = V^i_{t\_s}$ +
distance(A,B) * avg_cruise_speed)
$(mod_{fly} = cruise) \rightarrow (V^{i+1}_{capacity} = V^i_{capacity})$

References

[1] A. L. Blum and L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

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

[3] T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[4] A. Cesta and S. Fratini. The timeline representation framework as a planning and scheduling software development environment. In *The 28th Workshop of the UK PLANNING AND SCHEDULING*, 2009.

[5] A. J. Coles. Opportunistic branched plans to maximise utility in the presence of resource uncertainty. In *European Conference on Artificial Intelligence (ECAI)*, pages 252–257, 2012.

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

[7] M. E. desJardins, E. H. Durfee, C. L. Ortiz, and M. J. Wolverton. A Survey of Research in Distributed, Continual Planning. *AI Magazine*, 20(4), 1999.

[8] M. B. Do and S. Kambhampati. Solving planning-graph by compiling it into csp. In *Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 82–91, 2000.

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

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

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

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

[13] A. Gerevini, A. Saetti, and I. Serina. Case-based planning for problems with real-valued fluents: Kernel functions for effective plan retrieval. In *European Conference on Artificial Intelligence (ECAI)*, pages 348–353, 2012.

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

[15] M. Ghallab, D. Nau, and P. Traverso. The actor's view of automated planning and acting: A position paper. *Artificial Intelligence*, 208(0):1 – 17, 2014.

[16] M. Helmert. Decidability and undecidability results for planning with numerical state variables. In *Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 44–53, 2002.

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

[18] J. Hoffmann and R. I. Brafman. Contingent planning via heuristic forward search witn implicit belief states. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 71–80, 2005.

[19] S. Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, 1997.

[20] J. Kvarnström, F. Heintz, and P. Doherty. A temporal logic-based planning and execution monitoring system. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 198–205, 2008.

[21] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966.

[22] A. Lopez and F. Bacchus. Generalizing graphplan by formulating planning as a csp. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 954–960, 2003.

[23] C. L. López, S. J. Celorrio, and M. Helmert. Automating the evaluation of planning systems. *AI Commun.*, 26(4):331–354, 2013.

[24] R. Micalizio. Action failure recovery via model-based diagnosis and conformant planning. *Computational Intelligence*, page to appear, 2012.

[25] R. Micalizio, E. Scala, and P. Torasso. Intelligent supervision for robust plan execution. In *AI*IA 2011: Artificial Intelligence Around Man and Beyond*, volume 6934 of *LNCS*, pages 151–163. 2011.

[26] R. Micalizio, E. Scala, and P. Torasso. Towards robust execution of mission plans for planetary rovers. *Acta Futura*, 5:53–63, 2012.

[27] R. Micalizio and P. Torasso. Monitoring the execution of a multi-agent plan: Dealing with partial observability. In *European Conference on Artificial Intelligence (ECAI)*, pages 408–412, 2008.

[28] N. Muscettola. Hsts: Integrating planning and scheduling. Technical Report CMU-RI-TR-93-05, Robotics Institute, Pittsburgh, PA, March 1993.

[29] I. Musso, R. Micalizio, and E. Scala et al. Communication scheduling and plans revision for planetary rovers. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2010.

[30] H. Nakhost, J. Hoffmann, and M. Müller. Resource-constrained planning: A monte carlo random walk approach. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

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

[32] B. Nebel and J. Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76(1-2):427–454, 1995.

[33] T. A. Nguyen, M. Do, A. E. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, 190:1–31, October 2012.

[34] N. Policella, A. Cesta, A. Oddi, and S. Smith. From precedence constraint posting to partial order schedules: A CSP approach to robust scheduling. *AI Communications*, 20(3):163–180, August 2007.

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

[36] N. Roos and C. Witteveen. Diagnosis of plan execution and the executing agent. In *Lecture Notes in Artificial Intelligence (LNAI*, pages 357–366, 2005.

[37] F. Rossi, P. V. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[38] E. Scala. Numeric kernel for reasoning about plans involving numeric fluents. In *AI*IA 2013: Advances in Artificial Intelligence*, volume 8249 of *LNCS*, pages 263–275. 2013.

[39] E. Scala. *Reconfiguration and Replanning for robust Execution of Plans Involving Continous and Consumable Resources*. PhD thesis, Department of Computer Science - Turin, 2013.

[40] R. van der Krogt and M. de Weerdt. Plan repair as an extension of planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 161–170, 2005.