Plan Repair for Resource Constrained Tasks via Numeric Macro Actions

Enrico Scala
Dipartimento di Informatica - Universita' Di Torino
C.so Svizzera 185, Turin, Italy
scala@di.unito.it

Abstract

The paper addresses the problem of plan repair for tasks involving mandatory constraints on consumable and continuous resources, modeled as numeric fluents. The approach starts by proposing a new notion of numeric macro actions allowing to handle - as an extension to the classical macro action formulation - conditions and operations not only on the propositional fragment, but also on the numeric one. By reasoning directly on the current plan, the paper shows two techniques for selecting useful macro actions. Such macro actions, toghether with the original actions model, are then used by an off-the-shelf numeric planner for a faster plan repair task. To evaluate the techniques and the contribution of numeric macro actions, we experimented the approach on several numeric planning domains using Metric-FF as off-the-shelf planner. Results show that both strategies enhance the performance of the same planning system without macro actions. Even, one of the strategies turns out to be very competitive also with the specialized plan repair system LPG-ADAPT, both in terms of cpu-time, and stability of the repaired plan.

Introduction

The success of plan execution in real world domains can be several times compromised because of the occurrence of unexpected contingencies. Exogenous events, wrong assumptions on the state of the world or new goals to be achieved must be taken into account all along the task to accomplish.

Some uncertainty can be anticipated at planning time via conformant or contingent planning (Hoffmann and Brafman 2006; 2005), but the computational complexity of the arising planning problem would pose a severe bound on the applicability of these approaches in realistic scenarios; for instance conformant planning is EXPSPACE-complete, see (Haslum and Jonsson 2000).

As an alternative to these approaches, many authors suggest the continual planning paradigm (desJardins et al. 1999; Brenner and Nebel 2009; Ghallab, Nau, and Traverso 2014) as a viable and effective solution to the problem of plan execution in dynamic environments. Basically, in a continual planning approach an agent is allowed to interleave planning and execution all along the plan execution; in this way she can take into account unexpected situations (including new and/or different goals to achieve) once they actually arise. Of course, the efficiency of the replanning becomes crucial for reacting in a timely fashion. To this end, despite contrasting complexity results (Nebel and Koehler 1995), several approaches have adopted plan repair techniques, such as (Gerevini and Serina 2010; Garrido, C., and Onaindia 2010; van der Krogt R. and de Weerdt M. 2005; Cushing and Kambhampati 2005).

While such approaches provide an empirical evidence of the benefit of plan repair over replanning from scratch, the current methodologies are mainly focused on the maintenance of propositional conditions. Research on plans involving consumable and continuous resources has been mostly devoted to the problem of (off-line) plan generation (Hoffmann 2003; Do and Kambhampati 2003; Coles et al. 2012; Gerevini, Saetti, and Serina 2008). To the best of our knowledge, the only system supporting numeric fluents in a plan repair modality, and hence able to handle consumable and continuous resources (Fox and Long 2003), is the one reported in (Fox et al. 2006).

The main contribution of this paper is a new method for the plan repair problem involving continuous and consumable resources, modeled as numeric fluents. The approach grounds on the introduction of numeric macro actions. In extension to the classical macro action formulation (Botea et al. 2005), a numeric macro action captures not only propositional conditions, but also (i) the sufficient and necessary *numeric* conditions for the valid applicability of a sequence of actions, and (ii) how this sequence affects the numeric part of the current state (e.g., the impact on the resources). The paper will show how such aspects can be formulated as regular numeric preconditions and effects of PDDL 2.1 actions (Fox and Long 2003).

As previous approaches to the plan repair problem, we expect that, for the resolution of *small* deviations from the predicted trajectory (included unexpected resources consumption), the prior knowledge can benefit the efficiency of the plan repair task. The motivation of adopting a plan reuse step is even stronger considering all those scenarios where stability is a critical issue (Fox et al. 2006).

In this perspective, the paper presents two strategies for selecting useful numeric macro actions directly from the plan in execution. Both strategies generate an enhanced domain representation to be used in combination to an off-the-shelf numeric planner. Provided that the underlying planner used is complete, the enhancement does not prevent the completeness of the approach as also the original model of the actions is considered.

These strategies are implemented in a (planner independent) continual planning system handling dynamic environments and changing goals. To evaluate the mechanism, the strategies have been experimented over several numeric planning domains from the planning competition, and have been compared with LPG-ADAPT, the plan repair version of LPG (Gerevini, Saetti, and Serina 2008), and Metric-FF (Hoffmann 2003), which simulates a replanning from scratch methodology.

After a brief introduction on the formalism and the problem of reference, we present the method for building macro actions and these two plan repair strategies; then we describe how these strategies are implemented in a continual planning system. Finally, an experimental section is reported.

Motivating Example

To exemplify the mechanism that we propose in this paper, we consider a simple instance of problem from the numeric version of the *ZenoTravel* domain[1]. Assume to have 3 cities (`c1,c2,c3`) 2 airplane (`a1,a2`) and 2 passengers (`p1,p2`) and that our initial planning task involves as goal that the passenger `p1` must be in `c2`, and as initial state that all the airplanes and the passengers are in `c1`. A solution for this planning task is `(board a1 p1 c1) (fly a1 c1 c2) (debark a1 p1 c2)`. Note that, differently from the classical version, here actions encompass numeric conditions and effects. For instance, the `fly` action is possible only when there is enough fuel, and such fuel will be consumed after its execution.

Imagine that, just before the `board` starting, the user decides that also `p2` must be moved from `c1` to `c3`.

It is clear that the sub-plan for "moving passenger `p1` from `c1` to `c2`" is still feasible, despite the goal modification. The idea of this paper is to exploit this knowledge in form of macro action, and provide such a knowledge to the planner. In addition to previous works on macro actions, here we need to model resource profiles as well.

By using macro actions, we expect (i) an enhancement on the time performance of the planner, and (ii) that plans remain quite stable. Similarly to (Fox et al. 2006), we would like in fact that the system advantages situations exploiting past decisions, like for instance that the passenger `p1` takes the airplane `a1`.

Background

This section reports the reference planning language and the plan repair problem we are interested in. We assume the reader is familiar to the PDDL terminology (Fox and Long 2003).

[1]*Zenotravel* is one of the most widely employed benchmark domains in the international planning competitions. For details see http://ipc.icaps-conference.org/.

Basic Definitions

In line with the PDDL 2.1 formulation, we model consumable and continuous resources by means of numeric fluents. A numeric fluent is a special function mapping set of objects to a real value (e.g., `(fuel a1)`). Therefore, a world state has to specify each numeric fluent value of interest. Extending to the classical representation, we formally have:

Definition 1 (World State) *A world state is built upon a set F of propositions and a set X of numeric variables. Thus a state s is a pair* $< F(s), X(s) >$, *where F(s) is the set of atoms that are true in s (Closed World Assumption) and X(s) is an assignment in* $\mathbb{Q}$ *for each numeric variable in X.*

A world state in our formalism changes by means of numeric actions. Formally:

Definition 2 (Numeric Action) *Given F and X as defined above, a* numeric action *"a" is a pair* $< pre(a), eff(a) >$ *where:*

- *"*pre(a)*" is the applicability condition for "*a*"; it consists of:*
  - *a numeric part (*$pre_{num}(a)$*), i.e. a set of comparisons of the form (exp* $\{<,\leq,=,\geq,>\}$ *exp').*
  - *a propositional part (*$pre_{prop}(a)$*), i.e. a set of propositions defined over* F.
- *"*eff(a)*" is the effects set of* a*; it consists of:*
  - *a set of numeric operations (*$eff_{num}(a)$*) of the form (f,op,exp), where* $f \in X$ *is the* numeric fluent *affected by the operation op, which is one of* $\{+=,-=,=\}$.
  - *an "add" and a "delete" list (*$eff^{+}(a)$ *and* $eff^{-}(a)$*), which respectively formulate the propositions produced and deleted after the action execution*

Here, exp *and* exp' *are arithmetic expressions involving variables from* X. *An expression is recursively defined in terms of (i) a constant in* $\mathbb{Q}$ *(ii) a numeric fluent (iii) an arithmetical combination among* $\{+,*,/,-\}$ *of expressions*[2].

An action *a* is said to be applicable in a state *s* iff its propositional and numeric preconditions are satisfied by *s*. Meaning that (i) $pre_{prop}(a) \subseteq$ F(s) and (ii) $pre_{num}(a)$ must be satisfied (in the arithmetical sense) by X(s).

Given a state s and a numeric action a (such that a is applicable in s), the application of a in s, denoted by $s[a]$, (deterministically) produces a new state s' as follows. s' is initialized to be s; each atom present in $eff^{+}(a)$ is added to $F(s')$ (whether this is not already present); each atom present in $eff^{-}(a)$ is removed from $F(s')$; each numeric fluent f such that (f,op,exp) $\in eff_{num}(a)$ is modified according to the exp and the op involved.

The state obtained by a non applicable action is undefined. An undefined state does not satisfy any condition.

[2]For computational reasons, several numeric planners request such expressions to be linear (Hoffmann 2003; Coles et al. 2012). In our case, such a restriction is not necessary for the implementation of macro action. However, it becomes newly necessary once a numeric planner has to be used.

Having defined a state and the underlying state transition system given by the formulation of numeric actions, the following defines how a numeric plan can be obtained as a solution of a Numeric Planning Problem.

Definition 3 (Numeric Planning Problem) *A numeric planning problem Π is the tuple $< s_0, G, A >$ where, s_0 is the intial state, G[3] is a goal condition and A is a set of ground actions. A solution for Π is a total ordered set of n actions from A such that the (ordered) execution of these actions transforms s_0 into a state s_n where G is satisfied.*

As we are interested in reasoning about (sub)sequences of actions, given the formulation above, we allow the access to a segment of the plan by subscripting the plan symbol. More precisely, $\pi_{i \to j}$ with $i < j$ identifies the sub-plan starting from the i-th till the j-th action. Moreover when the right bound is omitted the length of the plan is assumed, i.e. $\pi_i \equiv \pi_{i \to |\pi|}$. Finally, we identify by $s[\pi]$ the state produced executing π starting from *s*.

Following the plan validity conditions defined in (Fritz and McIlraith 2007) and extended for the numeric setting in (Scala 2013), we consider the plan valid (and hence "executable") for a state s_i and a goal G, when the kernel associated to the suffix plan π_i is satisfied by s_i.

In real world dynamic environments, a plan has a very low chance to be valid till the end of the mission. There are many reasons why this could happen; there may be unexpected resources consumptions, some exogenous events may invalidate the current solution, the user may request a different set of goals. Assuming that, in order to monitor the current plan execution, the agent is capable of observing the environment where it is operating, similarly to the dynamic planning problem defined in (Fox et al. 2006), we can define a numeric plan repair problem (for an invalid plan) at the i-th step of the execution as follows:

Definition 4 (Numeric Plan Repair Problem) *A numeric plan repair problem Ψ is the tuple $< s_i, G', A, \pi_i >$ where s_i is the state observed after the execution of the (i-1)-th action from the plan π , G' is a new set of goals[4], A is the universe of actions we are considering, and π_i is the suffix of the plan still to be executed. A solution for Ψ is a totally ordered set of actions from A such that the execution of these actions brings the state s_i to a state satisfying the conditions expressed in G'.*

As highlighted in (Fox et al. 2006), even if some modifications could be necessary, in several scenarios it is requested that plans should remain as close as possible to the original formulation. For this reason, we consider optimal the plan solution for Ψ that minimizes the distance from the starting solution plan π_i. As metric for measuring this distance we exploit the action based distance defined in (Fox et al. 2006)[5]. That is, given two plans π and π' the distance $D(\pi, \pi')$ is the sum of the number of actions in π that are not in π' with the number of actions in π' but not in π.

[3]A goal condition is expressed as an action precondition.

[4]G' represents a variation w.r.t. the initial goal, where some predicate/comparison could be added/removed.

[5]This distance has been also exploited in (Nguyen et al. 2012) for finding "diverse" plans.

Numeric Macro Actions

The work by (Botea et al. 2005) presents a mechanism for the macro actions construction for classical planning. The approach relies on combining (incrementally) pairs of (successive) actions; the result is a macro action that keeps trace of the *progressive* effects of these actions with the *regressive* conditions that are necessary for executing such actions.

The construction mechanism reported in this section inherits this two basic steps, extending the work for handling also the numeric characteristic of the model of the action.

For clarity reasons, in the following we will focus just on the numeric aspect of the macro action creation[6]. In particular, given an actions sequence P, the objective is to infer:

- what are the numerical conditions for the execution of P, (e.g., the minimum amount of the power needed for the execution of a set of *fly* actions)
- what is the numeric contribution of executing P, (e.g., the overall amount of fuel consumed by the system after the execution of actions in P).

The crucial step refers to the way in which a pair of actions is combined; having defined this, it suffices to repeat the step as many actions the sequence of interest consists of. Algorithm 1 summarizes the procedure for the construction of a numeric macro action *c* starting from actions *a* and *b*.

As a first step, the routine unifies the parameters and the names of *a* and *b*. Doing so, we have a way to recognize the primitive actions the resulting macro action comprises. For instance, if we merge the `board (p1,a1,c1)` action with the `fly (a1,c1,c2)` action, then we will have the action `board_fly(p1,a1,c1,c2)`. Afterwards, the action must express which are the conditions for the applicability of *a* followed by *b*.

While the preconditions of *a* must be added - without any modification - in the set of preconditions of the macro action (line 6), we must take care somehow also of the preconditions expressed in the *b* model (lines 4 and 5).

To capture this aspect, the numeric preconditions of *b* have to be regressed toward *a* by keeping trace of all the numeric effects that *a* has on the variables involved in *b* preconditions[7]. This is achieved by the substitution of each numeric fluent involved in the expression reported in *b* (in particular in the comparison elements), with the numeric expression modeling how this fluent is modified by means of effects of *a* (line 5). For instance, if we have the pair (`refuel`, `fly`) we are sure that the fuel at disposal after the execution of the `refuel` corresponds to the capacity of the airplane. For this reason the arising condition for the `refuel_fly` will not involve a comparison on the fuel, but just on the airplane fuel

[6]For details on the propositional progression and regression mechanism, take a look at (Botea et al. 2005).

[7]The numeric regression mechanism here is similar to the numeric kernel generation for a plan. For details look at (Scala 2013), where it is showed that such mechanism effectively computes the weakest set of numeric conditions for the applicability of a plan.

capacity. Let us assume that `fly` requires 10 units of fuel. Since the `refuel` brings the fuel to the maximum capacity level (and in the ZenoTravel domain this is an invariant information of the problem), the fly action will not require any condition from the point of view of the macro action. As matter of facts the fluent `fuel` will be substituted with the fuel capacity of the airplane that, in case is greater than 10 unities, it always will be satisfied. Otherwise, the macro action is thrown away as it is an unfeasible actions sequence.

Note that the simplification and the possible removal should be done only once the macro is associated to a particular plan repair task. Doing so, a macro is able to capture variation also on numeric fluents, considered irrelevant (invariant) at planning time (e.g., the average consumption contributing in defining the applicability of the `fly` action in the *Zenotravel* domain may be different than expected).

The description for the propagation of the effects is very similar to the previous case, but in the opposite direction. Also in this case, the substitution mechanism is key in determining how each fluent is affected by means of the cumulative execution of *a* and *b*. Here, each expression involved in the numeric effects of action *b* (in particular the third term of the tuple (f, op, exp)), must take into account how the fluents taking part of the expression of the operator have been modified by *a*. As matter of facts, the generic fluent *f* represents the future while *exp* is evaluated in the state where the action has been applied. In the case of the macro action, this state is not represented explicitly (and this is one of the great advantage of the compression given by the macro actions); however, we can keep trace of the internal behavior of the action: this is what is moved ahead.

For instance, if we have an action modifying the fluent f by means of the operator += with the real value 5, we know that the action effect on this fluent is *f = f + 5*, i.e. $(f, +=, 5)$. Now if the action is merged with a previous action modifying the same fluent, and decreasing it according to the expression $g + 3$, we have to substitute the fluent *f* in exp (which is implicit in the increase operator), with the way it is modified *f - g - 3*. So, in PDDL terms, we can combine $(f, +=, 5)$ with $(f, -=, g+3)$ obtaining $(f, +=, 2-g)$ as new numeric effect of the resulting macro action. By doing so, the substitution mechanism is able to keep trace of the (possible several) interactions arising from (all) the numeric fluents involved in the merged actions (lines 8 to 10).

The point above is the only *tricky* part in the progression mechanism. In addition to this phase, we need to move forward also the effects of *a* on each fluent f such that f is not affected by *b* (lines 13 to 15), and numeric effects of *b* that do not have nothing to do w.r.t. the execution of *a* (line 12).

Plan Repair via Macro Actions

In literature, the utility of macro actions has been exploited in off-line plan generation for the purpose of providing shortcuts on the search space (Chrpa et al. 2013; Botea et al. 2005; Coles and Smith 2007). The main claim of this paper is that the macro action formulation can be even more beneficial in the task of repairing a plan[8].

Algorithm 1: Numeric Macro Action Construction

```
   Input: $a$ , $b$ - action
   Output: $ab$ - macro action
1  $name(ab) = name(a)\_name(b)$
2  $par(ab) = par(a) \cup par(b)$
   /* Precondition Constr. -- Regression   */
3  $pre_{prop}(ab) = (pre_{prop}(b) \setminus eff^{+}_{prop}(b)) \cup pre_{prop}(a)$
4  foreach $comp \in pre_{num}(b)$ with $f \in affected(eff_{num}(a))$ do
5      $pre_{num}(ab)$.add(subst(f,comp,$eff_{num}(a)$))
6  $pre_{num}(ab)$.addAll($pre_{num}(a)$)
   /* Effects Constr. -- Progression        */
7  $eff^{+}_{prop}(ab) = (eff^{+}_{prop}(a) \setminus eff^{-}_{prop}(b)) \cup eff^{+}_{prop}(b)$
8  foreach $numEff \in eff_{num}(b)$ do
9      if $numEff$ involves an $f : f \in affected(eff_{num}(a))$ then
10         $eff_{num}(ab)$.add(subst(f,numEff,$eff_{num}(a)$))
11     else
12         $eff_{num}(ab)$.add(numEff)
13 foreach $numEff \in eff_{num}(a)$ do
14     if $numEff$ refers to $f : f \notin affected(eff_{num}(b))$ then
15         $eff_{num}(ab)$.add(numEff)
16 return $ab$
```

Given a particular problem, the idea is that the plan to repair can be a rich source of knowledge for building macro actions. Such macro actions can be in fact built reasoning on all the subsequences of the actions the plan consists of. We expect in fact that a large part of the plan is still useful, and can be exploited during the search.

However, if we exhaustively pick up all the possible contiguous actions in a plan of length n, we will have to deal with $\frac{n(n-1)}{2}$ macro actions; this can become an issue for the branching factor of a planner.

In order to preserve only (hopefully) relevant macro actions, keeping out the ones which could represent useless additional information, in the next sections we propose two strategies: Suffix/Prefix Macro Actions (SPMA) and Causal Links Macro Actions (CLMA). Such strategies aim at trading off between considering or not all the possible macro actions given the plan at hand.

Suffix/Prefix Macro Actions

As we have observed in the formulation of a dynamic repair problem, the discrepancies encountered during the execution can refer to the current state and/or the goal. That is, the state where the remaining part of the plan has to be applied, and the conditions that the plan has to meet at the last plan execution state.

From this consideration, the first strategy we are going to propose is quite straightforward, but, as we will see in the experimental section, already effective. The idea is to create macro actions from the suffixes and the prefixes of the

[8]Of course, as anticipated, under the condition that the repair arises just because of a limited amount of discrepancies, or little changes in the goal set. This is somehow also justified by the results presented in (Liberatore 1998).

plan. Macro actions representing prefixes capture pieces of the plan (heuristically) easy to apply at the current state but that scarcely satisfy the goal set. On the other hand, macro actions representing suffixes capture sequences of actions that already achieve the goal set but that need extra actions at the beginning in order to become executable at some point.

To manage the (still large) amount of suffixes and prefixes to consider, the strategy is limited in picking up just $\frac{K}{2}$ macro-actions for each subset (prefix or suffix)[9]. In particular, we take the ones which are closer to the initial state (in terms of satisfaction of preconditions), and better support the goals set (in terms of satisfied goal conditions). More formally, given a macro action a, a state s_i and a goal G' we prioritize macros minimizing the following distance:

$$D(a, s_i, G') = |pre_p(a) \backslash F(s_i)| + |G'_p \backslash eff^+(a)| + |G'_p \cap eff^-(a)| + \sum_{c \in pre_{num}(a), s_i \not\models c} EucDist(X(s_i), c)$$

where the first three terms model the "propositional" distance, while the fourth approximates the numeric one. In the current SPMA implementation, we compute numeric distances only when c involves linear expressions. $EucDist$ represents the euclidean distance between the point associated with $X(s_i)$ and the hyperplane associated with c. As you can see from the formula, we take only the contribution of the constraints not satisfied by s. The overall measure is intended to provide an approximation on the effort the planner should spend to satisfy such conditions.

A limit of this strategy is that it tends to pick up very similar macro actions, since the selection of the macro does not consider previous selected macros, but just the relation between the current macro and the problem. To overcome this limitation, we propose an alternative strategy, which reasons about the plan and its internal (in)consistencies.

Causal Links Macro Actions

The idea at the basis of this strategy comes from the consideration that, during the plan execution, even if the plan is not consistent given the goal and the current world state, there may be sub-sequences of the plan that still preserve consistency in their internal structure. However, there are positions all along the plan that present inconsistencies. Such inconsistencies are actually open preconditions, also called missing causal links in the context of Partial Order Planning (POP) (Kambhampati 1997).

Taking inspiration from POP approaches, the "Causal Links Macro Action" reasons on the causal links structure arising from the plan repair problem.

In particular, the strategy saves each point where the inconsistency is detected, and uses such points to split the plan in a number of macro-actions. The idea is to preserve each subsequence of actions that is executable as a whole, once its (macro) preconditions are satisfied. Moreover, we add extra points for all those actions whose effects threaten the goal satisfaction. The procedure is described by algorithm 2.

[9]In the experiments, we empirically set $K = 6$ (3 prefixes and 3 suffixes).

Algorithm 2: Causal Links Macro Action

```
Input: $\pi_i$ - plan, $s_i$ - current state, $G'$ - goal
Input: $S$ - Macro Actions Set
1 begin
2     $S = \{\}$;
3     splittingPoints = {i,$|\pi|$};
4     foreach $j : a_j \in \pi_i$, $a_j$ has open preconditions or $a_{j-1}$ threatens $G'$ do
5         splittingPoints := splittingPoints $\cup$ {j}
6     foreach pair $s, e : s, e \in$ splittingPoints, $s < e$ and $\nexists k : s < k < e$ do
7         $S = S \cup \{build\_macro(\pi_i, s, e)\}$
8     return $S$
```

As a first step, the algorithm selects all the "inconsistency" points (the $splittingPoints$ set) by considering those actions whose preconditions are not satisfied or whose effects threaten the goal satisfaction (lines from 3 to 5). The missing preconditions come from (propositional/numeric) services that should be provided by the initial state, while the action threatening the goal can arise when the set of goal has been changed w.r.t. the original objectives. Afterwards, the procedure computes as many actions as there are fragments of "valid subplan". We expect that the real effort in the resolution of the arising planning task corresponds to overcome the multiple inconsistencies all along the plan. While these actions preserve the most of the computation done for solving some of the sub-problems of the overall planning task. As for the SPMA strategy, having the macro action model, the planner is informed of the requirements to apply these (sub)solutions, and of the (propositional and numeric) consequences of their application.

Running Example

In our motivating example, the two macro selection strategies presented so far would have acted as follows:

- SPMA: the suffixes and the prefixes selected are `(board a1 p1 c1)` `(fly a1 c1 c2)`, `(fly a1 c1 c2)` `(debark a1 p1 c2)`, and `(board a1 p1 c1)` `(fly a1 c1 c2)` `(debark a1 p1 c2)`, i.e., the whole plan. So the system produces three macro actions.
- CLMA: the only action produced is the one corresponding to the whole plan. As matter of facts, the only open condition is in the goal conditions set. In fact all the actions are still applicable but the passenger `p2` must be carried on in `c3`. So, the planner is allowed to combine the previous solution with the actions to bring `p2` at destination.

The domain can be hence enhanced with the knowledge of the previous plan. It is worth noting however, that SPMA have produced very similar sub-plans, while CLMA built only a macro action that is actually the most significant one.

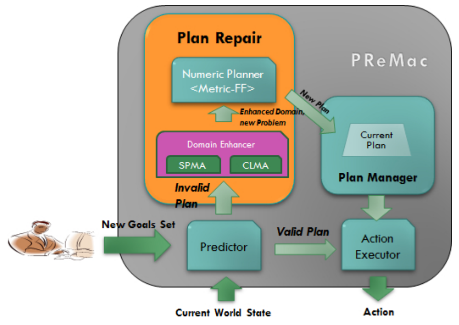

Figure 1: Continual Planning Architecture

As we will see in the experimental phase, this is key in determining the difference of performances between such two macro action generators.

Implementation and Evaluation

To evaluate the macro action notion, and the selection strategies built upon, we implemented a continual planner system. The architecture, Plan REpair via MACro actions (PReMac, see figure 1), is capable of monitoring the execution of the plan and triggering the plan repair task once such a plan becomes inconsistent w.r.t. the encountered observed state of the system, and the goal set. The plan repair is faced by the domain enhancer (i.e. either SPMA or CLMA) while Metric-FF (Hoffmann 2003) is used as numeric planner for the resolution of the arising planning task[10].

The main difference w.r.t. the most of continual planners[11] is the adoption of a preprocessing step just before the invocation of the planning tool. The module in charge of performing this operation is the domain enhancer anticipated above. This module takes in input the piece of the plan execute and produces macros according to the selected strategy. Such macros are then formalized in PDDL, but, differently from the original action schema set, they do not need any parameters. In fact, a macro action in our work is a ground actions sequence. In order to represent grounded predicates and numeric fluents, the domain model is enriched with a set of constants representing objects of the problem.

The system has been tested on 6 numeric domains of the International Planning Competition (IPC): *DriverLog*, *ZenoTravel*, *Hard ZenoTravel*, *Rovers*, *Depots*, and *Satellite*. To challenge the system in the management of resource constraints, we preferred domains where actions are preconditioned also in numeric terms, not only in the classical sense. As difference to standard domains we considered also a harder *ZenoTravel* where the refuel for airplanes is possible only in specific airports, and in the used *DriverLog*, we conditioned the `drive-truck` action to be executed only given a particular fuel threshold. To generate several instances of plan repair problems we used an approach similar to the one reported in (Fox et al. 2006). The suite of cases referring to the standard *ZenoTravel* comes directly from the benchmark made available by these authors.

In our experiments a test case corresponds to the tuple $< domain, problem, problem', plan >$: the *plan* is the solution of the planning task given by $< domain, problem >$. In the context of plan execution, the plan is actually the subplan of actions still to be executed, while *problem'* keeps trace of the current world state and the (possibly modified) goals set. *problem'* represents a slight variation of *problem*: the initial state or the goals can be different in order to simulate unexpected contingency, or changes in the goals set.

Cases have been in fact generated by randomly modifying the initial state and the goals set of *problem'*, both in terms of propositional atoms (mimicking unexpected action effects, or different assumptions on the world state) and in terms of resources amount at disposal (e.g. the fuel of the airplan). We aim at understanding the systems behavior when discrepancies affect propositional and resource aspects, in a crosswise fashion.

The generation of plan repair problems starts from the 10 most difficult problems of the benchmarks planning suite. For each of them, a starting plan has been generated (by using LPG), and a set of variant problems have been collected. Each variant differs from the starting problem up to 3 initial state information and 3 goals.

For comparison reasons, test cases have also ran with the LPG-ADAPT system, and Metric-FF used as replanning from scratch methodology. The performances of the systems are measured by taking into consideration the International Planning Competition metric[12] for the time, the coverage and the distance[13] score. Let S be a set of systems to be evaluated (SPMA, CLMA, LPG-ADAPT, and Metric-FF in our case), for each parameter p (time or distance or coverage), the score of a case for the system s is defined by means of $\frac{bestValue(p,S)}{value(p,s)}$. When p is time, value indicates the cpu-time spent for solving the repair task. The distance value is computed comparing the current plan and its repaired version and finally the coverage score is 1 when the case is solved, 0 otherwise. Note that when the case is not solved, also the other scores (time and distance) take 0. The total score for a domain is the sum of the scores obtained for each case in that domain.

Experiments ran on Ubuntu 10.04 with an Intel Core Duo@2.53GHz cpu and 4 GB of Ram. To emulate a plan execution in a real world scenario (where solution must be quickly provided), each computation has been allotted with

[10]The performance can be very different considering the particular search strategy employed in the planner. We adopted Metric-FF as it is the only able to handle actions with large preconditions and effects. Of course, modifications to the core of the planner could considerably increase the performances of the system.

[11]An exception is the work reported in (Garrido, C., and Onaindia 2010), where the Lama planning system (Richter and Westphal 2010) is used as black box for the plan repair problem. Knowledge compilation based approaches have received a considerable success also for other problem reformulation; e.g., (Taig and Brafman 2013; Albore, Palacios, and Geffner 2009)

[12]Experimented for several years in the planning community, the IPC metric is a well established technique for measuring planner performances. For details look at http://ipc.icaps-conference.org/

[13]This distance, also used in (Nguyen et al. 2012), has been firstly introduced in (Fox et al. 2006).

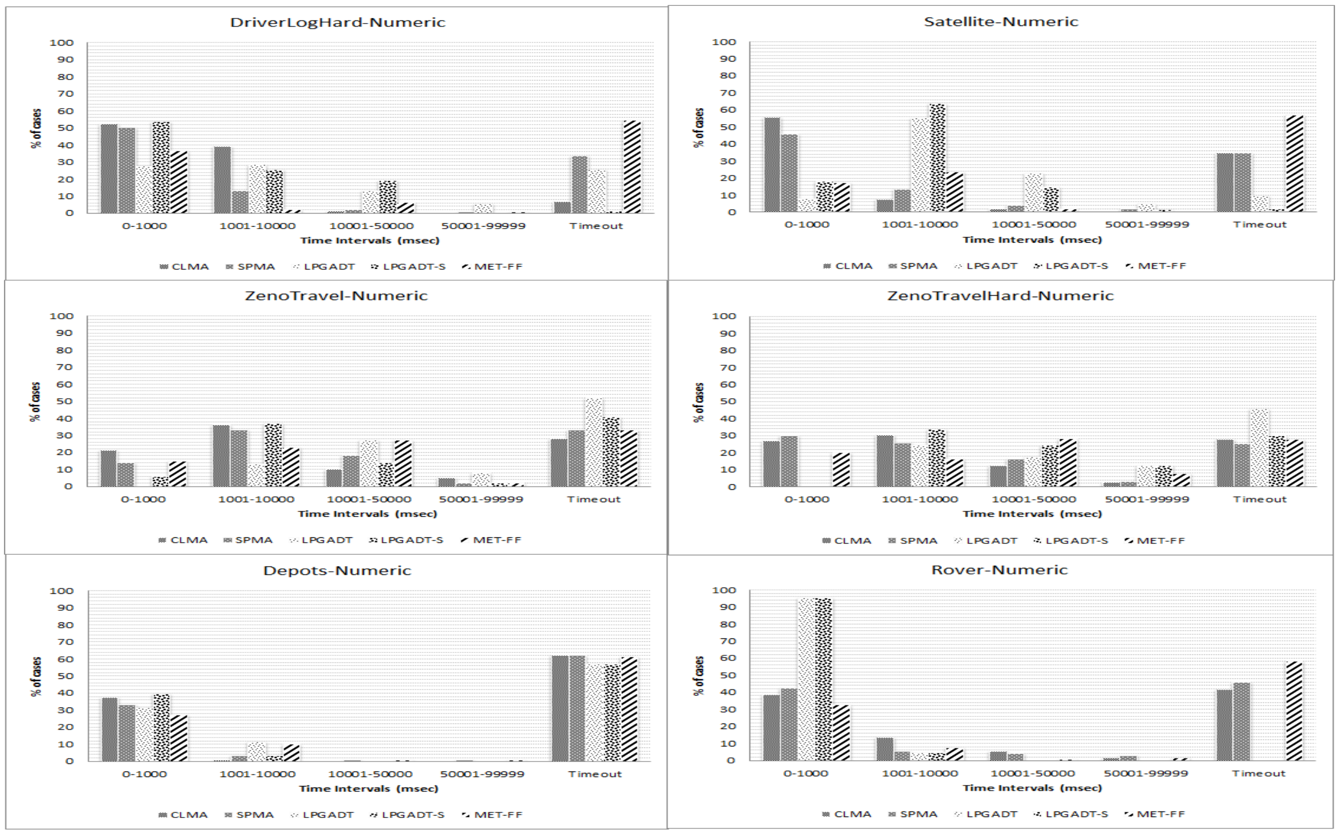

Figure 2: Percentage of cases solved in a given interval of *cpu-time* (msec), for each domain considered. CLMA, SPMA, LPGADT, LPGADT-S and MET-FF stands, respectively, for the two strategies proposed in this paper, the LPG-ADAPT system used with the quality and the speed (-S) setting. Metric-FF is the system playing the role of a replanner from scratch.

a maximum of 100 seconds of cpu-time.

Table shows the collected IPC results. In parenthesis the number of cases considered for each domain. Besides the strategies presented in this paper, LPGADJ, LPGADJ-S and MET-FF stand for LPG-ADAPT with the quality setting, LPG-ADAPT with the speed setting, and the Metric-FF with the original domain definition, respectively. Note that, for the strategies presented in this paper, the time-score includes also the cpu-time spent for macro actions preprocessing. However, such a time has been always negligible; the actual overhead of the strategies is due to the larger branching factor the planner has to cope with.

In all the experimented domains, the addition of the macro actions to the Metric-FF planning system turned out to be quite beneficial, for all the parameters tested. In particular, for the CLMA case, the advantage is quite large (408 vs 244) for the time score and also the coverage is increased. Of course, the distance score of Metric-FF w.r.t. all other systems is not comparable, as Metric-FF is not built with the concept of the stability, so the resulting plan could be also quite ”distant” from the initial formulation.

What is surprising in the experiments is that CLMA outperforms also LPG-ADAPT in its speed version, as far as it is concerned by the time score[14]. Making exception for the *Rover* domain, the time score of CLMA outperformed all the systems tested. The advantage is quite large for the two versions of the *Zenotravel* domain, while it is less prominent, but still significant for *Satellite* and *DriverLog*.

As refers the distance and the coverage score, LPG-ADAPT-S has been the best system used, even if the two plan repair strategies turned out to be very competitive in some domains (i.e., *Zenotravel* and *DriverLog*).

Finally, histograms reported in figure 2 provide an overview of the cpu-time spent by the systems, over all the tested domains. In particular, they show the percentage of cases solved for 5 intervals of time (0-1000,1001-10000,10001-50000,50001-99999,Timeout). Basically, the aim is to highlight some aspects of the time score presented in table , as the IPC scores are very coverage oriented.

Except for *Depots*, SPMA, but in particular CLMA are able to substantially decrease the number of timeouts of Metric-FF. In particular, in the *DriverLog* domain the CLMA strategy reduces the timeouts from 55 to 5. However, CLMA and SPMA often encountered more timeouts

[14]This was unexpected since CLMA is built to be planner independent, while LPG-ADAPT can take advantage from search control that are specific for the plan repair.

	Score	CLMA	SPMA	LPGAD	LPGAD-S	MET-FF
	Time	**94**	80	13	24	58
ZenotravelHard	Distance	**86**	38	60	**86**	16
(155)	Coverage	112	**116**	84	109	112
	Time	**37**	21	21	33	29
Depots	Distance	41	5	39	**49**	3
(121)	Coverage	46	46	**52**	**52**	47
	Time	**107**	67	10	81	52
DriverLog	Distance	107	41	50	**126**	10
(145)	Coverage	135	96	108	**143**	66
	Time	**63**	35	4	24	37
Zenotravel	Distance	34	12	42	**53**	8
(100)	Coverage	**72**	67	48	59	67
	Time	23	21	35	**137**	35
Rover	Distance	69	49	89	**119**	16
(151)	Coverage	88	82	139	**151**	63
	Time	**84**	69	21	52	33
Satellite	Distance	78	39	100	**128**	3
(149)	Coverage	97	97	135	**146**	64
	Time	**408**	293	104	349	244
Total	Distance	414	184	380	**562**	56
(821)	Coverage	550	504	566	**660**	419

Table 1: Overall Results considering the IPC Scores. Measures refer to the cpu-time, the distance and the coverage for each system, over all the tested domains.

than LPG-ADAPT (this is also evident by looking at the coverage score), and the reason why CLMA obtained a higher time score is that, when the task is solved, CLMA is very fast; otherwise the system is not able to provide an answer given the time deadline. This is evident by looking at the high percentage of cases solved rather efficiently (from 0 to 1 sec) by both CLMA and SPMA. Conversely, the LPG-ADAPT has a more stably behavior, and this is the reason of the advantage on the other two parameters tested.

Finally, it is also easy to see that CLMA performs better than SPMA in all the domains. In particular CLMA causes less timeouts, and in most of the domains solved more cases of SPMA in less than 1 sec. This means that the reasoning on the actual in-consistencies arising in the plan repair context (and hence on the affected causal links structure) is beneficial to select more informative macro actions.

Related Work

Several works have appeared in literature for the plan repair problem (Gerevini and Serina 2010; van der Krogt R. and de Weerdt M. 2005; Garrido, C., and Onaindia 2010; Brenner and Nebel 2009). Basically, these works have an underlying continual planning approach; actions are executed from the plan till this remains valid, in the sense defined by (Scala 2013) and (Fritz and McIlraith 2007), or the goal is achieved. As soon as an inconsistency is detected, a plan repair step is invoked. In particular, these works suggest that the plan repair can be more effective by exploiting a plan adaptation step. In this perspective a variety of techniques have been presented (Gerevini and Serina 2010; van der Krogt R. and de Weerdt M. 2005; Garrido, C., and Onaindia 2010). However, the most of them does not deal with consumable resources, but just focuses on the propositional fragment of planning problems. Resources have been dealt in the LPG-ADAPT system (Fox et al. 2006), where moreover the concept of plan stability has been introduced. Our work is able to manage consumable resources, but, differently from the LPG-ADAPT system, the methodology presented in this paper does not rely on a specific planner. Therefore, it can be adopted as a repair mechanism in any other planning systems. For the sake of evaluation, our strategy has been compared w.r.t. LPG-ADAPT.

In the more general context of robust execution, resources (in the limited form of the renewable case) have been managed in building (pro-actively) robust schedule (Policella et al. 2009). The robustness here is intended to provide tolerance on the time taken by actions. Similarly, temporal flexibility is proposed by the adoption of particular forms of Disjunctive Temporal Network (Conrad and Williams 2011). As innovation w.r.t. these methodologies, we deal with robust execution combining the reasoning on the resources, with the classical causal aspects of propositional planning. Moreover, since we model resources by means of numeric fluents, we do not require that such resources are just renewable; they can be in fact consumable, too.

The idea to combine two or more actions in a big unified structure is not new. Several approaches have been studied the adoption of macro actions in the off-line planning phase (Botea et al. 2005; Chrpa et al. 2013; Coles and Smith 2007). Basically the goal of these approaches is to supply the domain with additional action schemas encapsulating knowledge on the way actions can be combined each other.

In this paper we adopt macro actions for the plan repair problem. As we will see, differently to the plan generation phase, in our problem we do not need of restricting the macro generation to few primitive actions. This because we are mainly interested on grounded representations and hence our method does not suffer from the combinatorial explosion due to the increasing of the number of action parameters.

Finally, as an innovation to the current state of the art in the context of macro actions, we extended the structure to deal also with numeric information. Our construction mechanism does not need that predicates and numeric fluents are grounded, so the notion can be exploited also in off-line planning (Botea et al. 2005).

Conclusion

The paper presented an approach for the plan repair problem in presence of numeric fluents, which can appear in mandatory conditions to be satisfied for the applicability of actions, as well as for the reachability of the goal.

Firstly, the approach proposed the notion of numeric macro action; this structure extends the classical macro action (Botea et al. 2005; Coles and Smith 2007; Chrpa et al. 2013) for allowing to keep trace (both in the regression and progression mechanism) of the numeric information of an action. In addition to that, an effective method to build macro actions is introduced.

The numeric macro action formulation is essential for two plan repair strategies the paper proposes, namely SPMA and CLMA. Both strategies perform the repair in two steps: firstly, an enhanced version of the starting planning domain enriched with macro actions is created; then a numeric planner system off-the-shelf is invoked. The difference of these two strategies is in the way macro actions are selected. In particular, the strategy reasoning on the (possibly compromised) causal links structure of the plan to be repaired

(CLMA), turned out to be the most effective one. Experimental results showed that the strategy improves the planner performance, in the most of numeric tested domains. Even, the exhibited performances are competitive with the state of the art in plan repair, i.e. LPG-ADAPT (Fox et al. 2006).

As future work, we are planning an experimental phase with other planning systems, to understand which kinds of planner are suitable for macro actions. Intuitively, we expect that strategies should work well for planner where the branching factor is not a particular issue (e.g., Metric-FF); while they have scarce chance of success in optimal planners.

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *International Joint Conference on Artificial Intelligence (IJCAI-09)*, 1623–1628.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)* 24:581–621.

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

Chrpa, L.; Vallati, M.; McCluskey, T. L.; and Kitchin, D. E. 2013. Generating macro-operators by exploiting inner entanglements. In *Symposiium on Abstraction, Reformulation and Approximation (SARA-13)*.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research (JAIR)* 28:119–156.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. Colin: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

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

Cushing, W., and Kambhampati, S. 2005. Replanning: A new perspective. In *Poster Session in International Conference in Automated Planning and Scheduling (ICAPS-05)*.

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

Do, M. B., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research (JAIR)* 20:155–194.

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

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

Fritz, C., and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *Proc. of International Conference on Automated Planning and Scheduling (ICAPS-07)*, 144–151.

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

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

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

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

Haslum, P., and Jonsson, P. 2000. Some results on the complexity of planning with incomplete information. In *Recent Advances in AI Planning*. Springer. 308–318.

Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search witn implicit belief states. In *Proc. of International Conference on Automated Planning and Scheduling (ICAPS-05)*, 71–80.

Hoffmann, J., and Brafman, R. 2006. Conformant planning via heuristic forward search: A new approach. *Journal Artificial Intelligence Research (JAIR)* 170(6–7):507–541.

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

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

Liberatore, P. 1998. On non-conservative plan modification. In *European Conference on Artificial Intelligence (ECAI-98)*, 518–519.

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

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

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

Richter, S., and Westphal, M. 2010. The lama planner: guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39(1):127–177.

Scala, E. 2013. Numeric kernel for reasoning about plans involving numeric fluents. In et al., M. B., ed., *AI*IA 2013, LNAI 8249, Springer International Publishing Switzerland*.

Taig, R., and Brafman, R. I. 2013. Compiling conformant probabilistic planning problems into classical planning. In *Proc. of International Conference on Automated Planning and Scheduling (ICAPS-13)*.

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