

A general technique for plan repair

Marlene Arangú and Antonio Garrido and Eva Onaindia
Universidad Politécnica de Valencia
Departamento Sistemas Informáticos y Computación
Camino de Vera s/n. 46022 Valencia
{marangu, agarridot, onaindia}@dsic.upv.es

Abstract

In real world we have to deal with changing situations which may partially or entirely invalidate an executable plan. Current strategies for plan repair are basically aimed at solving problems where regular minor modifications in the initial or goal state occur in the plan. In this paper, we propose a new repair technique that identifies where the problem is located, which part of the plan needs to be repaired and it fixes the affected part of the plan. Our technique is capable to tackle any type of failure or modification in the plan.

1. Introduction

A plan is a set of actions that achieves some goals when they are applied on an initial situation. The execution of the plan will normally lead to the desired final situation if no unexpected events occur in the world and the resulting effects of the actions execution happen as planned. However, due to the changing nature of the real world, changes in either the initial state, the set of applicable actions and intermediate or final goals are very common and can invalidate the initial plan. When this happens there are two possible ways to reach the goal state: repairing the original plan, that is, changing it by adding/removing some actions so that it is still worth, or replanning, i.e. building a new plan from scratch.

Deciding a priori that, motivated by changes during the execution of the plan, it will be necessary to carry out a replanning process is not always viable. This can be due either to a lack of time or to commitments which can be acquired during the execution of the plan. It is also very possible that the new plan uses a large subset of the actions of the current plan and thus, replanning would result

in spending time to get something that which has already been achieved, which is not only contradictory, but can also cause unnecessary displeasure among organizations or agents which should not be affected by the replanning process.

Some authors assert that modifying an existing plan is sometimes harder than planning from scratch [10], but other authors consider plan repair as a better solution, like for example in the space missions of the NASA [9, 1], in multi-agent planning [12, 13, 11] and in domain-independent planning [4]. In [4] authors consider plan stability as a measure of the differences between the original plan and the new one in order to select the actions in the repair process. They obtain good results in different IPC (International Planning Competitions) domains, where the changes that force the plan repair are minimal -only in the initial or goal state. In [12, 13] authors work with the same type of changes as in [4], and they propose the use of planning strategies for the accomplishment of the repair. With the combination of two processes - refinement and unrefinement- they construct a removal tree that will be expanded until a solution is found (if there is one). The work of [11] also addresses the problem of repairing plans when unexpected events happen, looking for the next feasible action that allows reaching the goal.

This article presents a new general plan repair technique, for domain-independent planning, based on a combination of the plan repair ideas mentioned above: (a) changes that invalidate a plan are not only produced in the initial and goal states as indicated in [12] [13] [4], but they can also take place anywhere in the plan [11]; (b) it is possible that some initially valid actions are not longer valid, either the initial and goal state remain unchangeable or not; and (c) it is also possible that some of the actions of the original plan are still useful and they can be reused for the new plan.

2 Terminology and notation

By following standard notations and definitions in the literature [6], this section summarizes the basic definitions used in this paper.

Definition 1: Planning Problem. A planning problem is defined as a tuple $\langle \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$ where \mathcal{I} is the initial state, \mathcal{A} is a set of durative actions and \mathcal{G} represents the top-level goals of the problem. Time is modelled by means of \mathbb{R}^+ and the order relations “ $<$ ” and “ $>$ ” by chronological order. Typically, the domain of actions includes all actions in \mathcal{A} .

Definition 2: Durative Action. Let $a \in \mathcal{A}$ be a durative action whose start time is t_i and end time is t_j . The components that define a are a set of *conditions*, the action *duration* and a set of *effects* (as defined in the planning language PDDL2.2 [2]).

Definition 3: Causal Link. Let a_i and a_j be two durative actions $\in \mathcal{A}$. A causal link \mathcal{CL} , represented as $\langle a_i \xrightarrow{p} a_j \rangle$, denotes that a_i supports the condition p for a_j ; that is, $p \in \text{AddEff}(a_i) \wedge p \in \text{Cond}(a_j)$. a_i is said to be a producer action for a_j , and a_j is said to be a consumer action of a_i .

Definition 4: Plan. A plan $\Pi = \{a_1, a_2, \dots, a_n\}$ is a list of n durative actions, partially or totally ordered. A *valid plan* is a plan Π that is executable in the initial state \mathcal{I} and reaches the goal state \mathcal{G} .

Definition 5: Action Error. An error occurs when a condition $p \in \text{Cond}(a_i)$ is unsupported, i.e. $\nexists \langle a_k \xrightarrow{p} a_i \rangle$. Action a_i is called the failing action and we will refer to this action as a_ε from here on.

3 Proposed approach

Figure 1 depicts the architecture of our plan repair technique. The inputs of the system are a domain of actions \mathcal{A} , a problem P and the current plan Π . This technique works as follows:

Phase 1: Validation. The validation is an iterative process that, at each time point t_k , it verifies if the current state \mathcal{CS}_{t_k} satisfies the actions can begin, continue or end their execution. For those actions $\{a_i\}$ which begin or finalize at t_k , \mathcal{CS}_{t_k} is updated as $\forall a_i \mathcal{CS}_{t_k} = \mathcal{CS}_{t_k} \cup \text{AddEff}(a_i) - \text{DelEff}(a_i)$. The validation process stops either because all the actions in the plan have been satisfactorily verified or because a failure in the plan has been detected. In this latter case the algorithm proceeds with the second phase.

Phase 2: Repair. The repair process calculates the scope of the failing action a_ε in plan Π and selects a set of actions out of \mathcal{A} to fix the failure. The set of selected actions will be referred as Cand as they represent the set of *candidate* actions to be part of the final plan.

Algorithm 1 details the steps of the repair process. The core of the repair algorithm is an iterative procedure that

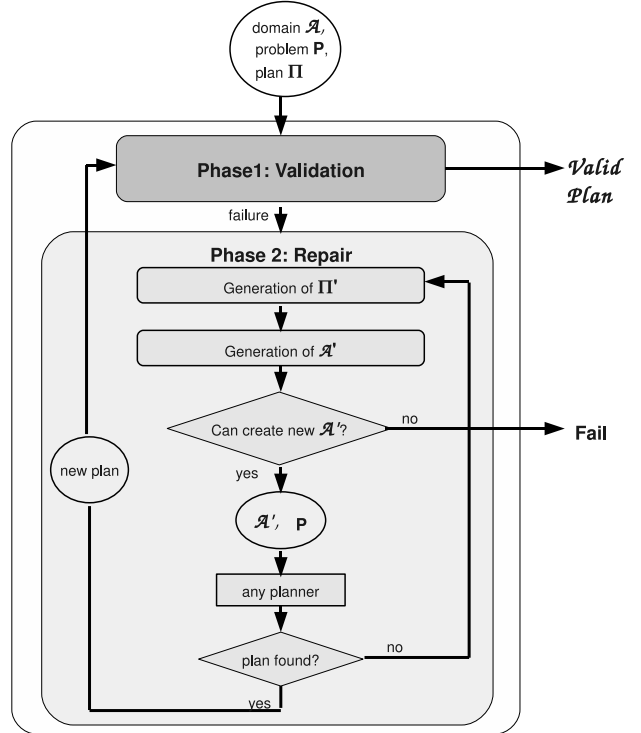


Figure 1. Schema of the general technique for Plan Repair

computes the set A' to solve Π' , i.e. the portion of Π which has been removed as affected by a_ε . Initially, Cand is initialized to a_ε . Then this set is augmented with the set of actions in Π involved with a_ε through a causal link plus the actions out of the domain A which might be potential producers for a_ε and potentials consumers of a_ε , that is the potential predecessor and successor actions of a_ε . In the next iterations, the set Cand is augmented with the actions involved with any action in Cand . Thus, the set Cand is composed of: i) the initial actions in Cand , ii) the potential immediate predecessor actions of the actions contained in Cand and iii) the potential immediate successor actions of the actions contained in Cand . In general, Cand is composed of the actions of Π which are affected by the failure plus the actions out of the domain A which might be potential solvers. The idea is to keep the affected actions in Π as candidates and leave the planner take the decision to re-use them or not to compute the final plan.

Following, we compute the set A' as $\Pi - \text{Cand} \cup \text{Cand}$, where $\Pi - \text{Cand}$ is the set of actions of Π which must *obligatory* be in the final plan, i.e. the set of actions which remain unaltered from the original plan. The set $A - A'$ will contain the unnecessary actions, i.e. the actions which will not be considered to solve the new problem. These actions will be pruned from the domain. Then we invoke the

Algorithm 1 Algorithm Repair

```
1:  $Cand = \{a_\varepsilon\}$ 
2:  $oldA' = \emptyset$ 
3: repeat
4:    $Cand = \{ \forall a_i \in \mathcal{A} : a_i \xrightarrow{f} a_j, \forall a_j \in \{Cand\} \} \cup$ 
       $\{ \forall a_k \in \mathcal{A} : a_j \xrightarrow{f} a_k, \forall a_j \in \{Cand\} \} \cup Cand$ 
5:    $A' = (\Pi - Cand) \cup Cand$ 
6:   if ( $A' \neq oldA'$ ) then
7:     run planner with  $A'$  and  $P$ 
8:     if plan_found then
9:       return Phase 1
10:    else
11:       $oldA' = A'$ 
12:    end if
13:  else
14:    FAIL
15:  end if
16: until plan_found or FAIL
```

planner with the problem P and the new domain A' (step 7). If the planner is able to find a plan, this plan is passed to the validator (step 9) to check the executability of its actions. Otherwise, the set A' is stored in variable $oldA'$, which is used to check any variation in the calculation of A' in step 5. In the next iteration, the process computes the new set $Cand$, which is now augmented with all the actions from Π and A involved with any action in $Cand$.

4 Experimental Results

The proposed technique has been implemented in Java and it has been tested in several domains from the last two planning competitions [8, 7] with two planners: LPG-TD 1.0 [5] and MIPS-XXL [3]. Two tests were made: in *Test 1* we made a change on an intermediate action (eliminating an action of the input plan to force the error); and, in *Test 2*, a change in \mathcal{I} . The tests were censored after 600 seconds.

Figure 2 shows the results obtained in the Storage-Time domain for Test 1, where the graphic a) shows the makespan of the resulting plan, and the graphic b) is the total computation time required to obtain the solution. As it can be observed, most of the times the makespan values are very similar in both our repair technique and replanning. However, the computation times in problems 6 to 20 are smaller when repairing with LPG-TD than when replanning with the same planner. This is a noticeable result if we take into account that the repairing process includes an initial fixed minimum simulation cost to detect the error in the input plan.

Variations in the makespan are mainly due to two fac-

tors: on the one hand, plan repair is trying to maintain a greater input plan stability like in [4]; and on the other hand, domain pruning is made on the basis of the found errors, whereas replanning only takes into account the global problem metric without domain pruning.

Test 2, shown in graphics c) and d) in Figure 2, was performed on the Satellite domain [7] using LPG-TD. In test 1, the same amount of time (10 minutes) was given for both repair and replanning but only the best solution was shown. However, in test 2, we also show the first solution found. As a consequence, concerning the quality (makespan) of the plan, differences between the first solution (less than 10 seconds) and the best solution (600 seconds) with replanning are quite significant; as less time worst quality. Equivalently, in graphic d) we can observe the first plan computed with the repair technique is less costly than the first plan computed with replanning. When using the repair technique, a second plan was only found in one of the problem instances (problem 5), while both makespan and CPU time remain stable.

Generating a plan from scratch is a very costly process; therefore, for repairing be more worthy than replanning, the time used in fixing the plan should be less than the time of computing a whole new plan. In some planners like LPG-TD, the more time is given, the better quality is obtained. For that reason, it is important to start out with a good-quality initial plan since this usually leads to modify only small parts of the plan and leave the rest unaltered, as it was initially planned. In these situations, repairing is less costly than replanning and it usually obtains better quality results.

5 Conclusions

The proposed repair technique systematically captures failures, delimits the scope of the failure, makes a domain pruning and allows reusing actions of the current plan. Additionally, this general technique is independent of both the planner and the domain.

The repair process has been compared with replanning in different domains and planners. Results show that the proposed repair technique achieves good results when the problem complexity grows, despite the fact that the repairing process has an initial minimal cost which is not present in the replanning process. This initial cost is due to the time spent in the validation of the input plan, the error detection and in the encoding of the new problem. Repairing also works in the line of plan stability as it tries to maintain the actions of the original plan that are not affected by the changes. This is an important issue since repairing is the technique that is commonly used to fix a problem during the execution of a plan and so plan stability is a valuable property, i.e. change the minimum part of the plan as possible to fix the problem and keep the rest unaltered.

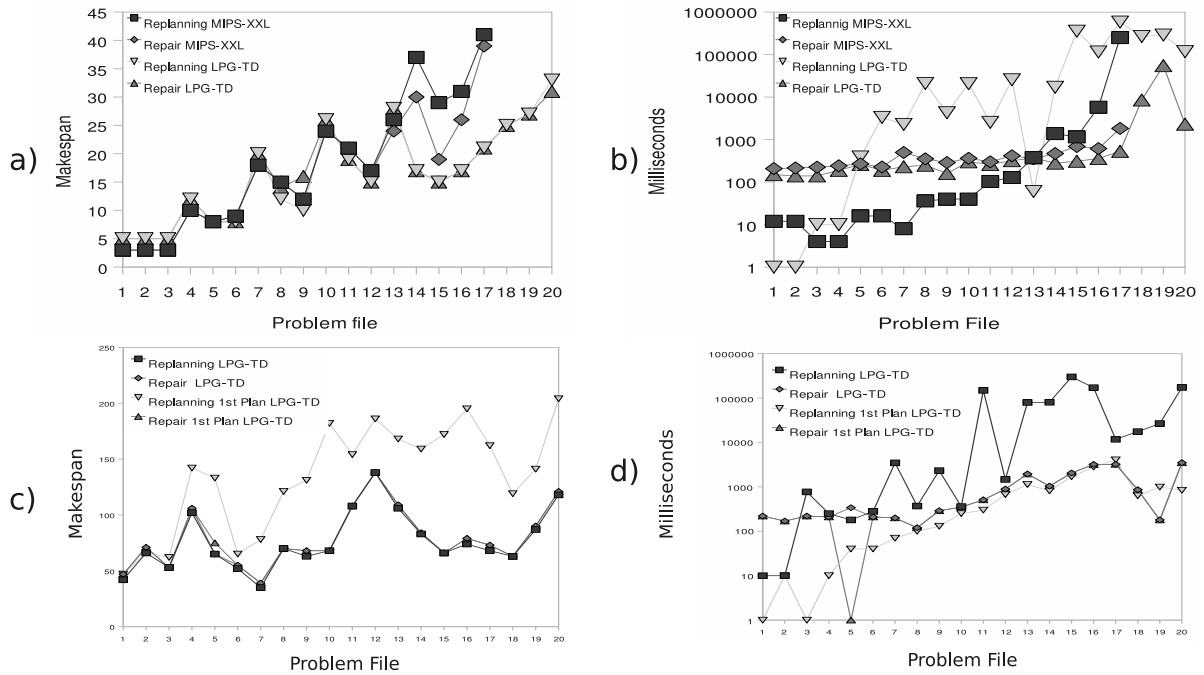


Figure 2. Replanning vs Repair. a) and b) belong to Test 1, Domain Storage-Time with LPG-TD and MIPS-XXL planners. c) and d) belong to Test 2, Domain Satellite with LPG-TD planner.

In the future, we would like to enhance the efficiency of our repair technique. We will design heuristics to better approximate the scope of the failure and prune actions from the domain. We also plan to extend our technique by using constraint programming (CSP) and to support more expressive modelling language features.

6 Acknowledgments

This work has been partially funded by Consolider Ingenio 2010 CSD2007-00022 project and by the Spanish Government TIN2005-08945-C06-06 project.

References

- [1] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proc. 5th ICAPS. Breckenridge, CO.*, April 2000.
- [2] S. Edelkamp and J. Hoffmann. PDDL2.2: the language for the classical part of IPC-4. In *Proc. ICAPS – International Planning Competition*, pages 2–6, 2004.
- [3] S. Edelkamp, S. Jabbar, and M. Nazih. Large-scale optimal pdl-3 planning with mips-xxl. In *Proc. ICAPS, 2006*.
- [4] M. Fox, A. Gerevini, D. Long, and I. Serina. Plan stability: Replanning versus plan repair. In *Proc. ICAPS*. AAAI Press, 2006.
- [5] A. Gerevini, A. Saetti, and I. Serina. An approach to temporal planning and scheduling in domains with predictable exogenous events. *Artificial Intelligence Research.*, 25:187–231, 2006.
- [6] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning. Theory and Practice*. 2004.
- [7] IPC. International planning competition, 2004. <http://ls5-www.cs.uni-dortmund.de/edelkamp/ipc-4/>.
- [8] IPC. International planning competition, 2006. <http://ipc.icaps-conference.org>.
- [9] N. Muscettola. *Intelligent Scheduling*, chapter HSTS: Integrating Planning and Scheduling. 1993.
- [10] B. Nebel and J. Koeler. Plan reuse versus plan generation: a complexity-theoretic perspective. *Artificial Intelligence*, 76:427–454, 1995.
- [11] O. Sapena and E. Onaindía. Execution, monitoring and replanning in dynamic environments. In *Workshop on On-Line Planning and Scheduling*, 2002.
- [12] R. van der Krogt and M. de Weerd. Plan repair as an extension of planning. In *ICAPS*, pages 161–170, 2005.
- [13] R. van der Krogt and M. de Weerd. The two faces of plan repair. In *BNAIC*, 2004.