PLAN REPAIR USING A PLAN LIBRARY

Roman van der Krogt and Mathijs de Weerdt

Delft University of Technology
{r.p.j.vanderkrogt | m.m.deweerdt}@ewi.tudelft.nl

Abstract

Plan library's have proven their added value to the efficiency of planning. In this paper, we present results on the use of a plan library to plan repair. We show that using a relatively simple library, we can already obtain significant improvements in efficiency compared to plan repair without a library.

1 Introduction

A planning problem is usually specified using a description of *(i)* the *current* (or *initial*) *state* the agent is in, *(ii)* the set of *actions* (together with their prerequisites and consequences) the agent is capable to perform, and *(iii)* the *goals* that the agent is aiming at, specified as a set of states. The *planning problem* then is to find the right sequence of actions leading the agents from the initial state to one of the desired states specified by the goals. Much effort has been put into developing efficient planning systems, as showcased at recent planning competitions [1, 14].

Planning alone, however, is not enough. Agents usually live in dynamic environments where goals may change or actions may fail. When an agent observes such a change it needs to update its plan. Techniques for updating a plan are called *plan repair* or replanning and have also been studied quite intensively over the past years (e.g. [13, 7, 3]).

We improve such a plan repair technique by reusing plan fragments that occur often in certain domains. These plan fragments are stored in a plan library and are used in the repair phase. The idea to use acquired knowledge was already proposed in [12], but only recently has it succesfully been applied to plan construction [4, 5]. In this paper we show its success in repairing plans.

The remainder of this paper is organised as follows. First, in Section 2, we describe the basic refinement replanning aproach that we have adopted. Then, we show how a plan library can be included in the search for a new plan during plan repair in Section 3. Section 4 describes our experiments and their results. Finally, Section 5 concludes the paper with a discussion of related work and a discussion of our results.

2 Refinement Replanning

The *refinement replanning* approach [13] is a general framework for plan repair algorithms. It is an extension of the *refinement planning* approach of Kambhampati [11] for planning algorithms. The main idea behind the refinement replanning approach is that plan repair consists of two phases (that can occur in any permutation, depending on the particular method): the first phase involves the removal of actions from the current partial plan that inhibit the plan from reaching its goals. The second phase is a regular planning phase, in which the partial plan is extended (refined) to satisfy the goals.

For example, suppose that we have a plan for driving to a meeting by car. Then, upon walking to the car we see that one of its tyres is flat. A simple repair for this plan could be to add actions that change the tyre with a spare one, and drive to the meeting as planned. Now suppose this is a very important meeting at which you do not wish to come late. In that case, replacing the tyre could cost too much time. Instead, it would be better to remove the drive action from the plan, and to replace it with actions using a taxi for transportation. Thus, to repair a plan, a planner should not only employ a *refinement* strategy for extending the plan with actions that will reach the goals

(such as replacing the tyres in the example). Planners should also employ an *unrefinement* strategy to consider removing actions from a plan that are obstructing a proper solution (such as the drive actions in the previous example).

It is important that these refinement and unrefinement strategies are tuned to one another. By taking each others peculiarities into account, they can efficiently work together. Fortunately, there exists a general unrefinement heuristic that can be automatically tuned to a chosen refinement strategy [13]. The idea is to use the refinement heuristic (i.e. a planning heuristic) as part of an unrefinement heuristic (i.e. for plan repair). First, the unrefinement heuristic calculates a number of possible unrefinements of P (i.e. ways in which actions can be removed from the plan). Of course, not all possible unrefinements can be considered, for these are far too many. Instead, a concept called *removal trees* is used to select a polynomial amount of unrefinements that are to be considered. Each such removal tree shows which actions can be removed from the plan if a certain set of initial conditions is no longer to be used, or if a certain set of goals is no longer required. Thus, one could think of them as "what-if" scenarios: what if we would not execute these actions, would we be able to efficiently reach the goals in another way, at the same time repairing our plan? For each of those scenarios, we can use a planning heuristic to estimate the amount of work it will take to complete this deteriorated plan into a plan that is a solution.

The benefit of this heuristic is that we automatically get an unrefinement heuristic that works together with the chosen refinement heuristic. This means that using this method, we can easily derive a system that adds plan repair capabilities to existing planners. This has the additional benefit that the method can be easily upgraded when new and more efficient planning heuristics are devised. Experiments have shown that the POPR plan repair system, which employs the above heuristic, is competitive with existing plan repair systems [13].

3 Introducing a Plan Library

Given the refinement replanning framework, there are two possible locations to add support for a plan library: the refinement strategy and the unrefinement strategy. We have chosen to add support for a plan library in the refinement strategy. There are two reasons for this. First, we expect that using a plan library helps to improve the speed, and analysis of the performance of the POPR system that we adapted shows that the majority of time is spent in the refinement strategy. Secondly, by adapting the refinement strategy (and its heuristic function), we implicitly include support for a plan library in the unrefinement phase, as this phase re-uses the refinement heuristic.

In order to minimally change the behaviour of the POPR system, we have chosen to work with *macro-actions* (in the terminology of [4, 5]). A macro-action is not just one action, but represents a whole plan from the library. From the outside a macro-action has the same properties as a normal (atomic) action. Its preconditions equal the preconditions of the plan it represents and so does its effects. During refinement, we do not work directly with plans, but instead use the special macro-actions. For most of the time, these macro-actions are treated as regular actions: they can be instantiated, inserted into a plan, their preconditions have to be satisfied and their effects can be used by other actions. However, as soon as a macro-action is fully instantiated, has it preconditions satisfied, and is neither threatened by nor threatening another link, it is replaced by the plan it represents: the macro-action is removed and the corresponding plan is retrieved from the plan library and inserted. Next, all ordering constraints on the macro-action are applied to the replacement actions and the search continues.

This method of dealing with plans has an important benefit over *(i)* directly include plans from the library in the main plan during search, and *(ii)* replacing the macro-actions when the complete plan is computed. The benefit over the first option (directly working with plans) is efficiency. By resolving all issues (such as instantiation, open conditions and threats) first, the system can efficiently deal with the action, after which it translates the constraints on the action to constraints on the added plan. This way, it has to find the constraints only once, instead of for each of the actions of which the added plan consists.

The disadvantage of the second option (replacing macro-actions at the end of the planning cycle) is that plan quality degrades, while in our solution we can reuse parts of plans that have been added in an earlier stage. Hence, the quality of our solutions is better.

4 Experimental Results

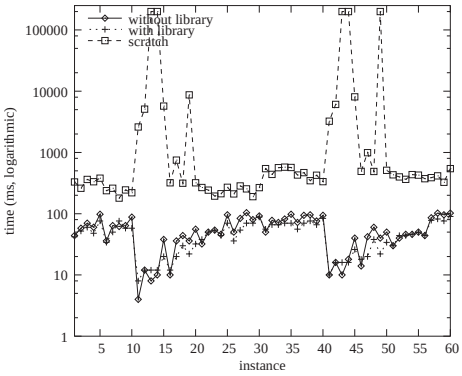
For the experimental validation of our technique, we added a plan library to the POPR plan repair system [13]. As a benchmark set, we used the standard set as proposed by [8]. This set consists of over 250 plan repair problems in three different domains: Gripper, Logistics and Rocket. The problems can be divided into 7 sets (2 each for the gripper and rocket domains, and 3 for logistics). Each set contains variants on the same test problem, each with a few changes to the initial state or the goals. For example, the gripper domain features a robot equipped with two grippers. It can `move` through a number of rooms, and has to move balls from their current location to another (using `pick` and `drop` actions). Examples of modifications in this domain are: "ball 2 is located in room B instead of in room A", or "ball 5 should no longer be brought to A, but to C".

For each of the domains, we created a single macro-operator that captures some domain knowledge. For example, in the gripper domain this is an action that represents the following sequence: a `pick`-action to pick up a ball, then a `move` to move to another room and finally a `drop` to drop the ball there. This encodes the behaviour that is required to move a ball from one location to another. Similar macro-operators were added to the other domains. This plan was specified as a macro action as follows:

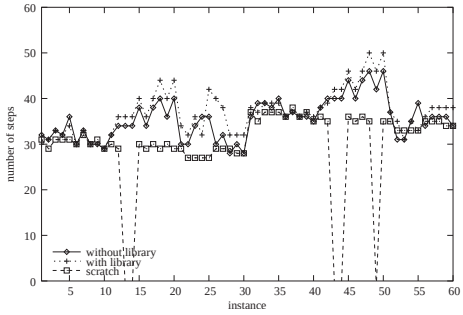
```
(:action macro-pick-move-drop
 :parameters (?from ?to ?obj ?gripper)
 :precondition (and (room ?from) (room ?to) (ball ?obj)
                (gripper ?gripper)(at-robby ?from) (at ?obj ?from)
                (free ?gripper))
 :effect (at ?obj ?to))
```

Figure 1 shows a number of representative graphs of our results. First, Figures 1.a and 1.b show the runtime and plan quality (the number of actions in the plan; lower is more efficient, thus better) in the Gripper domain. Instances 1-30 form one benchmark set, as do instances 31-60. (For reference, the results of planning from scratch are also included.) From the first figure, we can observe that plan repair using a plan library is slightly (but significantly, see the table at the end of this section) faster. The second figure shows that this increase in performance comes at a price: plan quality is degraded. This is caused by the fact that sometimes not the complete plan that is represented by the macro action is required, but only a part of it. However, it may be easier to find a solution using the macro action, than by including the required actions themselves. Further investigation shows that the search space is also significantly smaller, i.e. the number of plans that are considered is less. This can be seen in Figure 1.c. However, the search space is reduced by almost 40%, while the runtime only decreases by only 16%. So, fewer plans are considered, but the cost of expanding macro-actions into plan parts offsets the possible gains to some extent.

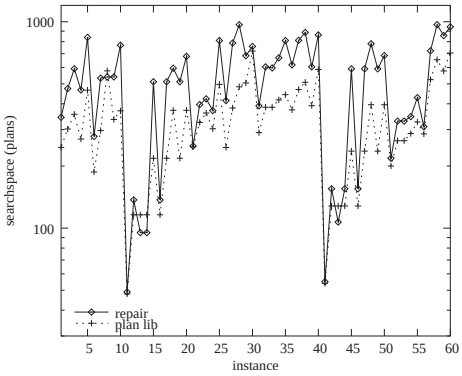
Results for the Logistics and Rocket domains are similar. Figures 1.d through 1.f show some results from the Logistics benchmarks. Again, we see a increase in performance due to improved searchspace statistics. Plan quality suffers less in this domain, however, as can be seen in the table below. The last two graphs, 1.g and 1.h, show results of the Rocket domain (again for both sets: instances 1-30 form one set, 31-60 the other). Again we see a significant improvement in peformance. In this domain, the plans produced are not significantly different when we compare plan produced with and without a library. The table below summarises our results, comparing plan repair with and without a plan library. It also shows the significance results from a pairwise t-test on the data.



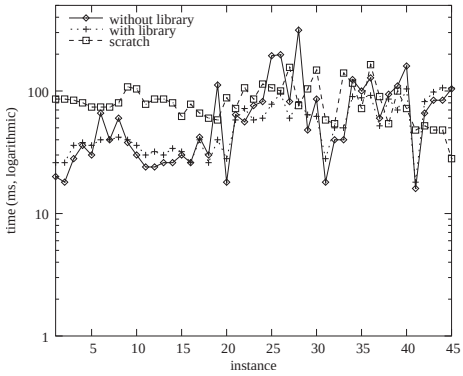
a. Gripper (performance)



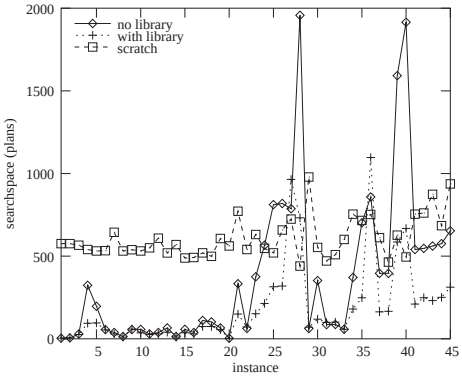
b. Gripper (quality)



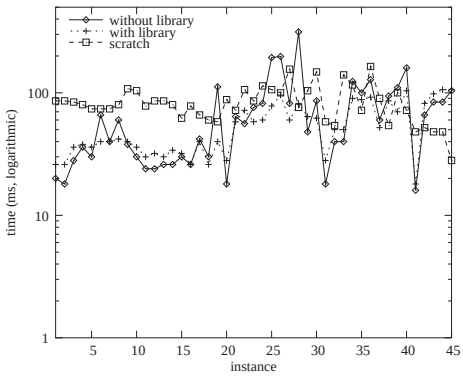
c. Gripper (searchspace)



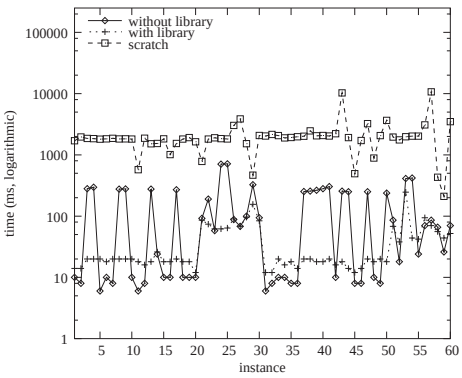
d. Logistics-A (time)



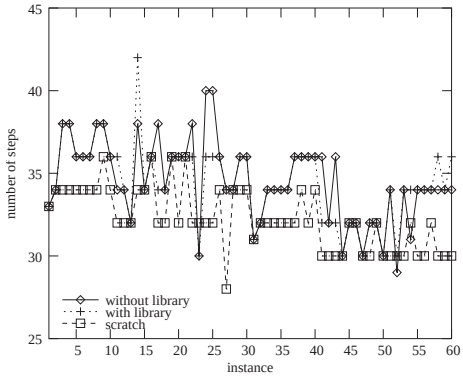
e. Logistics-A (search space)



f. Logistics-B (time)



g. Rocket (time)



h. Rocket (plan quality)

Figure 1: Results

		without library	with library	t	p
Grip	time	$\mu = 55.7, \sigma = 29.0$	$\mu = 46.5, \sigma = 23.4$	6.4	< 0.01
	plan size	$\mu = 35.5, \sigma = 4.4$	$\mu = 37.0, \sigma = 4.9$	-4.6	< 0.01
	nodes gen.	$\mu = 512.4, \sigma = 257.5$	$\mu = 330.0, \sigma = 157.9$	10.0	< 0.01
	nodes vis.	$\mu = 100.7, \sigma = 65.6$	$\mu = 61.7, \sigma = 51.1$	6.8	< 0.01
Log-A	time	$\mu = 78.1, \sigma = 56.3$	$\mu = 72.6, \sigma = 42.9$	1.4	**0.17**
	plan size	$\mu = 60.9, \sigma = 5.4$	$\mu = 59.6, \sigma = 4.9$	3.0	< 0.01
	nodes gen.	$\mu = 372.6, \sigma = 476.4$	$\mu = 189.3, \sigma = 247.7$	3.9	< 0.01
	nodes vis.	$\mu = 206.2, \sigma = 283.5$	$\mu = 94.1, \sigma = 154.2$	3.8	< 0.01
Log-B	time	$\mu = 70.0, \sigma = 58.6$	$\mu = 55.3, \sigma = 26.2$	2.2	< 0.04
	plan size	$\mu = 47.4, \sigma = 4.6$	$\mu = 47.4, \sigma = 4.4$	0.07	**0.94**
	nodes gen.	$\mu = 435.1, \sigma = 580.5$	$\mu = 142.0, \sigma = 130.2$	4.0	< 0.01
	nodes vis.	$\mu = 204.9, \sigma = 282.7$	$\mu = 57.6, \sigma = 56.5$	4.0	< 0.01
Log-C	time	$\mu = 89.5, \sigma = 67.4$	$\mu = 75.9, \sigma = 38.9$	2.2	< 0.03
	plan size	$\mu = 57.1, \sigma = 4.8$	$\mu = 56.6, \sigma = 4.7$	2.0	**0.06**
	nodes gen.	$\mu = 502.2, \sigma = 610.3$	$\mu = 186.7, \sigma = 186.0$	4.6	< 0.01
	nodes vis.	$\mu = 234.2, \sigma = 283.0$	$\mu = 76.5, \sigma = 80.4$	4.9	< 0.01
Rock	time	$\mu = 133.4, \sigma = 165.3$	$\mu = 38.9, \sigma = 41.3$	4.7	< 0.01
	plan size	$\mu = 34.5, \sigma = 2.5$	$\mu = 34.4, \sigma = 2.4$	0.7	**0.47**
	nodes gen.	$\mu = 1913.3, \sigma = 2504.3$	$\mu = 233.0, \sigma = 397.5$	5.3	< 0.01
	nodes vis.	$\mu = 886.0, \sigma = 1437.9$	$\mu = 70.7, \sigma = 151.3$	4.5	< 0.01

5 Discussion

Planning and plan repair is usually perceived as a one-shot process, i.e. without taking into account previous experience. However, in practice, planning is often performed repeatedly. Case-based planning [9] can be employed to make use of past experience. In this approach, the solutions to previous planning problems are stored. When a new planning problem is to be solved, the solution to a similar problem can be retrieved from memory and adapted. However, as we noticed before, it is often beneficial not to store complete plans, but rather *parts of plans* that are often used [12]. Recently, a number of planners have started incorporating such *macro actions* (i.e. actions that correspond to a number of basic actions executed in sequence). Macro-FF [4] is based on the FF planner by Hoffmann [10]. Macro-FF learns a set of macro actions through observing a number of sample problems of a domain. For these problems, it builds a planning graph [2] and examines which actions can occur in combination. For each of those combinations, it determines the usefulness by solving the problem with and without a macro-action encoding this combination. Macro-operators that reduce the planning effort (measured in the number of nodes expanded during search) are kept. These macro-operators are used as a heuristic for exploring the search space more efficiently when searching for a plan.

Marvin [5] employs macro actions by creating smaller instances of a problem using (almost) symmetry identification [6]. For symmetrical objects, it is likely that a similar subplan can be used to satisfy the goals on those objects. For each class of objects, one smaller problem instance is produced and its solution is used to produce macro-actions. Furthermore, if Marvin encounters plateaux in the search space (areas where a whole set of neighbouring states have the same heuristic value, where it is difficult to determine the best direction to continue the search), the system "memoises" action sequences that lead from the start of a plateau to a state that is strictly better. These plateaux-escaping macro-actions can later be used to escape similar plateaux more quickly. Like Macro-FF, Marvin uses the macro-actions as a heuristic. In addition, some of the macro actions are added to the list of actions that may be used during planning, like in our approach. However, in Marvin the macro-actions are not replaced with the list of basic actions they represent until the final plan.

Our approach is different in two aspects from the two planners employing macros. Firstly, our emphasis has been to show that macro-operators are useful in the context of plan repair. Hence, we did not include a domain analysis that learns a plan library for us. Rather, in our current system we have chosen to provide the library explicitly. Secondly, our way of including macro actions in the

search differs from both Macro-FF and Marvin. Whereas Macro-FF does not include macro-actions at all in a plan (but only uses them as a heuristic) and Marvin includes macro-action as a whole, our system replaces a completely instantiated macro-action with the actions it represents. Although we have not yet thoroughly investigated the effect of this decision in isolation, we conjecture that it has a positive effect on the size of the plans produced, because it allows to reuse part of a macro-action when possible.

Our experiments focussed on the difference between regular plan repair and plan repair using a plan library. The results differ slighly across domains, but in general we see that employing a plan library leads to a significant decrease in the size of the search space, and hence an improvement in performance (also statistically signficant). The result on the quality (size) of plans is a slight degradation: in some cases the plans produced using a library are worse, but in others they are not significantly different.

In the future, we intend to investigate the use of the plan library in the unrefinement phase. During the computation of removal trees, we can already take into account which actions form a macro action. In addition, we would like to explore the construction of a plan library that is both efficient for the refinement phase and the unrefinement phase.

References

[1] IPC-4: International planning competition 2004. `http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/`.

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

[3] G. Boella and R. Damiano. A replanning algorithm for a reactive agent architecture. In *Artificial Intelligence: Methodology, Systems, and Applications (LLNCS 2443)*, pages 183–192. Springer Verlag, 2002.

[4] A. Botea, M. Müller, and J. Schaeffer. Using component abstraction for automatic generation of macro-actions. In *Proceedings of the International Conference on Automatic Planning and Scheduling ICAPS-04*, pages 181 – 190, Whistler, Canada, June 2004.

[5] A. Coles and A. Smith. Marvin: Macro-actions from reduced versions of the instance. In *IPC4 Booklet, ICAPS 2004*, June 2004. Extended Abstract.

[6] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 956–961, 1999.

[7] A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proc. of the Fifth Int. Conf. on AI Planning Systems (AIPS-00)*, pages 112–121, Menlo Park, CA, 2000. AAAI Press.

[8] A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proc. of the Fifth Int. Conf. on AI Planning Systems*, pages 112–121, 2000.

[9] K. J. Hammond. Case-based planning: A framework for planning from experience. *Cognitive Science*, 14(3):385–443, 1990.

[10] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of AI Research*, 14:253–302, 2001.

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

[12] R. van der Krogt, A. Bos, and C. Witteveen. Plan fragment libraries. In *Proceedings of the Thirteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC-01)*, pages 399–406, 2001.

[13] R. van der Krogt and M. de Weerdt. Plan repair as an extension of planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-05)*, Monterey, California, 2005.

[14] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *Journal of AI Research*, 20:1–59, 2003.