In: Beiträge zum 10. Workshop *Planen und Konfigurieren (PuK-96)*

Complete Eager Replay

Héctor Muñoz-Avila and Frank Weberskirch
University of Kaiserslautern, Dept. of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-mail: {munioz|weberski}@informatik.uni-kl.de

Abstract

We present an algorithm for completely replaying previous problem solving experiences for plan-space planners. In our approach not only the solution trace is replayed, but also the explanations of failed attempts made by the first-principle planner. In this way, the capability of refitting previous solutions into new problems is improved.

1 Introduction

Case-based planning has shown to speedup the general problem solving process [16, 7, 8, 2, 6, 4]. For solving a new problem selected cases are reused in order to minimize the search effort of the first-principle problem solver. Different approaches have been developed for reusing previous problem-solving experiences. These approaches are motivated by different factors, which can be schematized as follows:

Base level planner: Classical planners have been divided into plan-space and state-space planners based on the space where search is performed. The former show a better performance in domains for which the plans are nonserializable [1]. State-space planners perform better in domains with an intrinsic linearity [17]. Algorithms to reuse cases have been introduced for both types of planners, plan-space [7, 8, 6, 4] and state-space planners [16, 2].[1]

Interleaving case-based and first-principle planning: There are systems that interleave case-based planning with first-principle planning [16, 4], whereas others first perform case-based and then first-principle planning [7]. In general, it is difficult to decide when to interleave, although some general heuristics have been proposed [3].

Degree of repair of previous solutions: Some algorithms perform a complex repair strategy [8, 6], while others leave the refitting effort to the first-principle planner [7].

[1] In [9] a case-based planner has been introduced whose base-level planner is a deductive reasoner. Thus, it does not fit into the division between plan-space and state-space.

Eager replay [7] has been shown to be an effective technique for reusing previous problem-solving experiences for plan-space planners. In eager replay, solution traces from cases are replayed when solving a new problem and a partial solution (i.e., a plan with open goals or unsolved threats) is obtained. This partial solution is then completed by first-principle planning. Thus, no interleaving of case-based and first-principle planning is done. An untreated limitation of eager replay is the degree of repair, as the repair effort is totally left to the first-principle planner. Thus, a large amount of planning effort may be needed to complete the partial solution.

We present an algorithm that extends the eager replay scheme in order to overcome its low degree of repair. In our approach, the solution trace of a previously selected case also is replayed. However, our algorithm additionally replays explanations of failed attempts to obtain the solution, made by the base level planner when the case was solved. In this way, the *complete* problem solving experience is replayed and can be used by the base level planner to avoid failing decisions.

2 Motivation

Different techniques have been investigated for ensuring the relevance of the retrieved cases, ranging from using domain-independent techniques [16, 8] to domain-specific techniques [15], passing by mixed approaches [11]. However, it is not realistic to suppose that relevant cases will always be found. There are two reasons for this: (1) There is no case in the case base that entirely "fits" into the new situtation, (2) retrieval cannot take much time in order to balance the trade-off condition between search effort and reuse effort [16, 5].

In this situation *complete eager replay* shows its strengths. Figure 1 compares complete eager replay to eager replay. A case has been selected for replay and we suppose that the decision labeled B cannot be replayed in the new situation. Further, we suppose that the decision labeled A needs to be rejected to complete

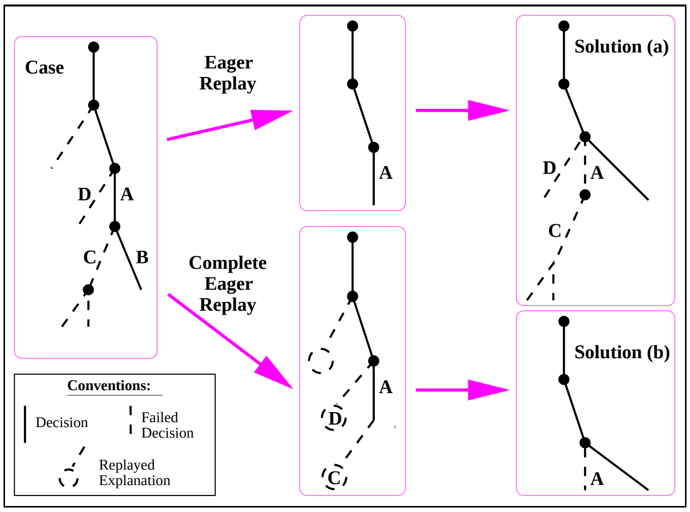

Figure 1: Completion after (a) eager replay and (b) complete eager replay of a case

the solution of the new problem. Solution (a) sketches the search path that must be followed when completing the plan obtained in eager replay. Notice that during completion some of the failed attemps made in the case are repeated, for example the decisions C and D. In complete eager replay, explanations of failed attempts are replayed too. As a result, unnecessary backtracking because of known failures is avoided during the completion process (see solution (b)).

We will now explain in more detail, how our algorithm automatically reconstructs justifications stored in the cases during replay.

3 The Base Level Planner CAPLAN

CAPLAN [18] is the SNLP-like [10, 1] base level planner of CAPLAN/CBC for generative planning. It uses a partially ordered plan representation (S, O, B) consisting of plan steps S, ordering constraints O among the steps and variable binding constraints B. At the beginning of the planning process, the set of open preconditions is constituted by the goals of the problem. The planning process proceeds by making decisions about the *establishment of open preconditions* and the *resolution of threats*. The former is achieved by adding a causal link between a new or an existing step and the step that consumes the condition, the latter is achieved by adding ordering or binding constraints that remove the threat. The planning process terminates if no open preconditions are left and all threats to existing causal links are resolved. Both, precondition establishment and threat resolution, are *choice points* of the planning algorithm.[2] Making wrong decisions causes backtracking and decreases the performance of the planning system. Supporting CAPLAN in this decision-making process is the purpose of the case-based control component CBC.

For representing knowledge about plans and contingencies that occur during planning, CAPLAN is built on the generic REDUX architecture [13, 14]. Key concepts of REDUX are goals, constraints, and contingencies. Planning proceeds by applying operators to goals, what may result in subgoals and in assignments (figure 2.a). Applying an operator is called a *decision* and represents a backtracking point as different operators might be applicable to a goal. Assignments originally

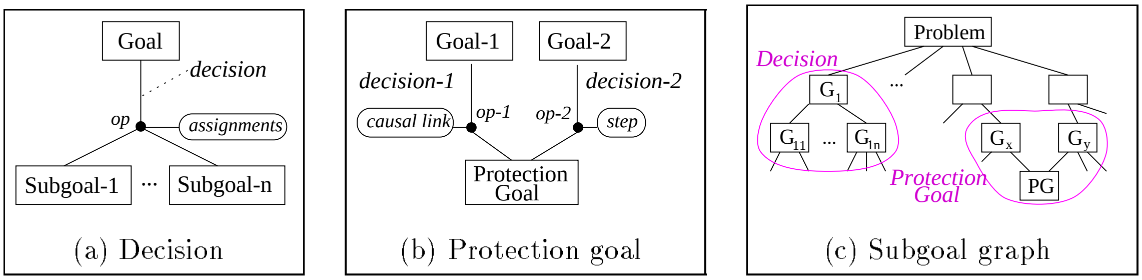

Figure 2: The subgoal graph

[2]Whether to establish preconditions or to resolve threats can be seen as another choice point with the difference that it is no backtracking point and, thus, unimportant for replay.

are thought to assign values to variables, more generally, they stand for modifications made in the plan (addition of steps/orderings/constraints). So, the mapping of SNLP concepts to REDUX concepts is straightforward. There are (1) goals to establish open conditions (precondition goals) and (2) goals to resolve threats (protection goals). Each refinement method of SNLP is represented as a class of operators that can be chosen to be applied to such a goal.

Goals and subgoals build the *subgoal graph* (figure 2.c). It represents basic dependencies between goals and subgoal as well as between subgoals and decisions. Originally, REDUX makes the assumption that each goal can have only one parent goal. This is not adequate for SNLP as protection goals represent threats and must depend on two goals, first, the goal with an operator that added the threatened causal link, second, the goal with an operator that added the threatening step (figure 2.b). REDUX has been extended for CAPLAN here. This extension of the dependency structure is important for automatically identifying threats and threat resolutions that become invalid after the rejection of a decision [18]. We omit further details, as they are beyond the purpose of this paper.

4 Complete Decision Replay in CAPLAN/CBC

A case in CAPLAN/CBC consists of a structure that encapsulates the successful decisions made at the choice points during a problem solving episode. Additionally, a case contains information concerning the rejection justifications of failed decisions. More concrete, CAPLAN/CBC stores the following elements in a case: (i) the subgoal graph, (ii) for each goal the chosen decision (applied operator) and the failed decisions, (iii) for each decision all rejection justifications. These justifications are reasons for rejecting a decision at a certain choice point [18].

The rationale behind storing justifications in cases is that decisions taken during the replay process may have to be rejected in order to complete the plan (e.g., decision A in figure 1 had to be rejected to obtain a solution). The case-based control component CBC guides the generative planner CAPLAN using this information about successful and failed decisions. As a result, reconstructing rejection justifications from the case will avoid persuing alternatives which are known to be invalid from the case.

Replaying a case consists of two phases: (1) *reconstruction of the subgoal graph* (i.e. subgoals and selected operators recursively as complete as possible) with respect to the current problem, (2) *reconstruction of justifications* for the rejection of operators. After a case C has been chosen to be used in a new problem situation, the goals of the problem are mapped to corresponding top level goals of the case C. Let G be the set of matching goals of the problem. Figure 3 sketches the algorithm for replaying the decisions of the case C for the goals G with repect to a current plan P. N is an initially empty list of goals that could not be matched immediately and so are delayed (see below).

Phase 1 (**ReplaySubgoalGraph**), the main part of the algorithm, reconstructs for each goal that matches a goal of the case, the corresponding part of the subgoal

ReplaySubgoalGraph(P, G, N, C)
For each goal $g \in G$:
- compute $cs = conflictSet(g)$
- get goal g_c of case C that matches g
- match cs and $conflictSet(g_c)$
- get chosen decision d_c of g_c from case
- IF $\exists op \in cs$ with $matches(op, d_c)$ THEN
· apply operator op to goal g
· get subgoals of goal g
· match subgoals of g and subgoals of g_c
· let SG be the set of matching subgoals
· ReplaySubgoalGraph(P, SG, N, C)
· ReplayDelayedGoals(P, N, C)
- ELSE
add goal g to the set N

ReconstructJustifications(P, C)
- get set D_C of reconstructed decisions
- for each justification j_c of a $d_c \in D_C$:
IF $\exists dependentDecisions(j_c)$ THEN
· create justification j matching j_c
· add j to decision d

ReplayDelayedGoals(P, N, C)
For each goal $g \in N$:
- compute $cs = conflictSet(g)$
- get goal g_c of case C that matches g
- match cs and $conflictSet(g_c)$
- get chosen decision d_c of g_c from case
- IF $\exists op \in cs$ with $matches(op, d_c)$ THEN
apply operator op to goal g (modifies P)

Figure 3: The replay algorithm of CAPlan/CbC

graph. The computed conflict set of a goal is always a set of operators that can be applied to the goal. If the matching operator for a decision of the case is not yet in the conflict set of the goal, the replay of this decision is delayed and this goal is added to the list N of goals that could not be linked to a goal of the case. This can for example happen with a phantom operator[3] that uses a step which has not yet been replayed in the current plan. It will be delayed until the needed step is added by the replay procedure, or it is skipped if the step is not created during the replay. Otherwise, the same decision as in the case is replayed, i.e. the same operator is applied to the goal, and the algorithm is called recursively for subgoals that matched subgoals in the case. After the recursive call **ReplayDelayedGoals** makes a new try to replay the goals that were delayed before.

Phase 2 of the replay (**ReconstructJustifications**) starts after the reconstruction of the subgoal graph. First, all decisions of the case that were reconstructed are collected. This are all successful or failed decisions of the case for which a corresponding operator was found in the conflict set of some goal in the current problem. For each rejection justification of such a decision the algorithm checks whether a reconstruction of the justification is possible. This is the case if all decisions that are referred to in the justification exist in the actual situation. If this test is positive an equivalent justification for the decision in the current plan is added. Otherwise, the justification can be skipped as it depends on the existence of a part of the case that has not be reconstructed.

The first phase is comparable to what eager replay does: it follows the derivation path stored in the case in form of the subgoal graph and selects the same alternative at every decision point for which it finds information in the case. The second phase is new: it replays experiences about failed decisions of the case and so avoids that the planner has to find out the already known reasons for failure again.

[3] *Phantom operators* represent simple establishment (they don't add a new step) [18].

5 Results and Conclusions

To measure the effectiveness of the replay procedure presented here, we performed the following experiment in the domain of process planning [12]: A series of problems were solved by the base level planner CAPlan and the plans were in the case base. While solving the problems we misdirected CAPlan at key choice points to ensure that justifications for wrong decisions which result in backtracking at those key choice points were constructed. The created case base was then used to solve new planning problems (variations of the cases). For each problem a case was selected and replayed with and without reconstructing justifications of failed decisions. Table 1 summarizes the results. The second and third column show the averages of the

Replay	**Complete**	**Normal**
Total Time (s)	25	65
#Inferences	37	129
#Decisions	33	72
#Valid Decisions	21	21

Table 1: Effectiveness of complete decision replay.

measurements made during the completion process for all given problems when complete replay or normal replay (i.e. without reconstruction of justifications) was used. We measured the total completion time, the number of inference steps to find the solution, the total number of decisions made during completion and the number of valid decisions. The ratio between the number of interence steps and the number of valid decision shows the degree of backtracking needed to find the solution. Our results show that complete eager replay leads to less backtracking. The overhead caused by reconstructing the justifications was approximately 25% of the replay time, which was smaller than the completion time (not shown in table 1). However, table 1 shows the advantage of reconstructing justifications over replaying only valid decisions as in [7]: completion time decreased more than 50% and significantly less inference steps were needed to find a solution. Nevertheless, The overhead of using this algorithm is approximately two times over usual replay (i.e., without calling the procedure *ReconstructJustifications(P, C)*). Deciding, when to reconstruct the justifications is still subject of research.

References

[1] A. Barrett and D.S. Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[2] S. Bhansali and M.T. Harandi. When (not) to use derivational analogy: Lessons learned using apu. In D.W. Aha, editor, *Proceeding of AAAI-94 Workshop: Case-based Reasoning*, 1994.

[3] B. Blumenthal and B. Polster. Analysis and empirical studies of derivational analogy. *Artificial Intelligence*, 1994.

[4] A. G. Jr. Francis and A. Rahm. A domain-independent algorithm for multi-plan adaptation and merging in least-commitment planning. In D. Aha and A. Rahm, editors, *AAAI Fall Symposium: Adaptation of Knowledge Reuse*, Menlo Park, CA, 1995. AAAI Press.

[5] A.G. Jr. Francis and A. Rahm. A comparative utility analysis of case-based reasoning and control-rule learning systems. In *Proceedings ECML-95*, number 912 in Lecture Notes in Artificial Intelligence, 1995.

[6] S. Hanks and D. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligece Research*, 2, 1995.

[7] L. Ihrig and S. Kambhampati. Derivational replay for partial-order planning. In *Proceedings of AAAI-94*, pages 116–125, 1994.

[8] S. Kambhampati. Expoiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*, 10(2):213–244, 1994.

[9] J. Koehler. Flexible plan reuse in a formal framework. In *Current Trends in AI Planning*, pages 171–184. IOS Press, Amsterdam, Washington, Tokio, 1994.

[10] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634–639, 1991.

[11] H. Muñoz-Avila and J. Hüllen. Retrieving relevant cases by using goal dependencies. In M. Veloso and A. Aamodt, editors, *Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR-95)*, number 1010 in Lecture Notes in Artificial Intelligence. Springer, 1995.

[12] J. Paulokat and S. Wess. Planning for machining workpieces with a partial-order nonlinear planner. In Y. Gil and M. Veloso, editors, *AAAI-Working Notes 'Planning and Learning: On To Real Applications'*, New Orleans, 1994.

[13] Ch. Petrie. *Planning and Replanning with Reason Maintenance*. PhD thesis, University of Texas at Austin, CS Dept., 1991.

[14] Ch. Petrie. Constrained decision revision. In *Proceedings of AAAI-92*, pages 393–400, 1992.

[15] B. Smyth and M.T. Keane. Retrieving adaptable cases. In M.M. Richter, S. Wess, K.-D. Althoff, and F. Maurer, editors, *Proceedings of the 1st European Workshop on Case-Based Reasoning (EWCBR-93)*, number 837 in LNAI. Springer, 1994.

[16] M. Veloso. *Planning and learning by analogical reasoning*. Number 886 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1994.

[17] M. Veloso and J. Blythe. Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of the 2nd International Conference on AI Planning Systems (AIPS-94)*, pages 13–19, 1994.

[18] F. Weberskirch. Combining SNLP-like planning and dependency-maintenance. Technical Report LSA-95-10E, Centre for Learning Systems and Applications, University of Kaiserslautern, Germany, 1995.