Efficient Plan Adaptation through Replanning Windows and Heuristic Goals

Alfonso E. Gerevini[1] and Ivan Serina[2]

[1] DEA, Università degli Studi di Brescia, via Branze 38, 25123 Brescia, Italy
gerevini@ing.unibs.it

[2] Free University of Bozen – Bolzano, Viale Stazione 16, 39042 Bressanone, Italy
ivan.serina@unibz.it

Abstract

Fast plan adaptation is important in many AI-applications. From a theoretical point of view, in the worst case adapting an existing plan to solve a new problem is no more efficient than a complete regeneration of the plan. However, in practice plan adaptation can be much more efficient than plan generation, especially when the adapted plan can be obtained by performing a limited amount of changes to the original plan. In this paper, we investigate a domain-independent method for plan adaptation that modifies the original plan by replanning within limited temporal windows containing portions of the plan that need to be revised. Each window is associated with a particular replanning subproblem that contains some "heuristic goals" facilitating the plan adaptation, and that can be solved using different planning methods. An experimental analysis shows that, in practice, adapting a given plan for solving a new problem using our techniques can be much more efficient than replanning from scratch.

1 Introduction

Fast plan adaptation is important in many AI-applications requiring a plan reasoning module. A typical plan adaptation task consists of modifying a previously generated plan in order to use it for solving a new problem which is "similar" to the original one, in the sense that only a few facts in the initial and goal states are different. This process can be either *off-line* (e.g., adapting a plan retrieved from a plan library before its execution), or *on-line* (e.g., adapting a plan during a "mixed-initiative" construction of it [3], or during its execution).

From a theoretical point of view, in the worst case adapting an existing plan is no more efficient than a complete regeneration of the plan [15]. However, in practice, we expect that in many cases plan adaptation should be much easier than a complete replanning, because the adapted plan can be obtained by performing a limited amount of changes to the original plan. Our general goal is the development of plan adaptation methods that perform efficiently especially in such cases.

ADJ is a domain-independent planning system based on planning graphs [1], that uses a collection of systematic and heuristic search techniques for solving plan adaptation tasks by replanning certain portions of the original plan.[1] In this context, a plan is

[1] ADJ is available by inquiring the authors.

Proceedings of the 15th International RCRA workshop (RCRA 2008):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Udine, Italy, 12–13 December 2008

a partially ordered set of STRIPS-actions, where each action is associated with a time step and unordered actions are associated with the same time step (indicating that these actions can be executed in any relative order). A first version of ADJ using the IPP planner [13] was presented in [8]. In this paper, we propose a significant extension of the original version of ADJ based on new heuristics for fast replanning, and we investigate the use of a different style of planning for the replanning phase. Moreover, we introduce a plan optimization phase in ADJ, and we present new experimental results evaluating our approach.

ADJ is based on identifying and resolving the inconsistencies (with respect to the new problem) that are present in the original (input) plan by revising limited portions of the plan. An inconsistency is an action-precondition that is not satisfied, a goal of the new problem that is not achieved, or a pair of mutually exclusive actions [1]. The portions of the plan that are revised are delimited by some temporal windows on the plan containing inconsistencies at specific time steps. Each replanning window is associated with a particular replanning (sub)problem that is solved using a systematic or heuristic search algorithm (in our implementation we used the search algorithms of IPP [13] and FF [10], but other algorithms could be used as well). During the adaptation process a temporal window can be incrementally enlarged up to contain, in the worst case, the full plan. In such a case a complete replanning is performed. When a subplan for solving the replanning problem associated with a temporal window W is computed, the actions of the plan under adaptation that are in W are replaced with the new subplan. This technique is complete, in the sense that if the replanning algorithm is sound and complete and the new problem is solvable, then the revision of the input plan for the old problem leads to a correct plan for the new problem.

In the rest of the paper, first we give a general description of our plan-adaptation method, followed by a more detailed description; then we present the results of an experimental evaluation showing that, in practice, our approach can be much more efficient than replanning from scratch.

2 Plan Adaptation, Replanning Windows and Heuristic Goals

In this section we describe our plan adaptation technique. We assume that the reader is familiar with the Graphplan-style of planning [1].

2.1 A General Description of ADJUST-PLAN

Given a plan π_0 for a planning problem Π_0 and a new problem Π, differing from Π_0 in some initial or goal fact(s), the plan adjustment process of ADJ consists of three main phases:

1. Analysis of the input plan to determine a subset of the actions in π_0 that are *applicable* for solving Π.

2. Identification of the set S of *inconsistencies* that are present in π_0 with respect to Π (an inconsistency is a pair of mutually exclusive actions, an action with some unachieved precondition(s), or a goal of Π which is not achieved).
3. Revision of π_0 to repair S obtaining a valid plan for Π.

The first phase is performed by mapping each action of π_0 to an action node of the planning graph $\mathcal{G}$ for the *new* problem (if such a node exists). This mapping considers three cases: (a) the number of time steps involved in π_0 is the same as the number of levels in $\mathcal{G}$; (b) the number of time steps involved in π_0 is higher than the number of levels in $\mathcal{G}$; (c) the number of the time steps involved in π_0 is smaller than the number of levels in $\mathcal{G}$. Each of these cases will be discussed in the last part of Section 2.

The second and the third phases of the adaptation process are accomplished by an algorithm, ADJUST-PLAN, that is correct and complete: the adjusted plan is a valid plan for Π, and if there exists a plan for Π, then an adjusted plan is computed. Figure 1 describes the main steps of ADJUST-PLAN, while some important additional details are given in the next subsections. The first step of ADJUST-PLAN identifies and removes from π_0 the actions that are not applicable in the context of the new problem. Then step 2 identifies the earliest time step at which π_0 contains an inconsistency. Since we are considering deterministic domains, this can be accomplished in polynomial time by simulating the execution of the plan of π_0 (analogously, the facts that are necessarily true at any time step can be determined in polynomial time).

Then ADJUST-PLAN processes the time step `i` identified at step 2, trying to remove the inconsistencies by replanning from time `i - 1` to time `i`. If there exists no plan, or a certain search limit is exceeded, then the replanning window is enlarged (e.g., we replan from `i - 1` to `i + 1`). The process is iterated until either a (sub)plan is found, the window has been enlarged up to include all the actions of the plan and the (complete) replanning has failed, or the search has reached a predefined CPU-time limit (`max-adjust-time`). Note that in our current implementation of ADJUST-PLAN, during replanning (step 5) the actions of π that are present in the replanning window are ignored (a new planning graph for the replanning problem is constructed). The replanning process at step 5 is performed by first constructing the corresponding *re*planning graph, and then searching it by using the backtracking scheme of IPP or FF.

At step 6 of ADJUST-PLAN the replanning window can be increased going either backward in time (i.e., `init-time` is decreased), forward in time (i.e., `goal-time` is increased), or both. Enlarging a replanning window corresponds to revising a larger portion of the plan under adaptation. Such a portion will be replaced by the subplan solving the replanning problem associated with the enlarged window (when this is found). ADJ has a default value for `max-adjust-time` that can be modified by the user. In principle, if `max-adjust-time` is set to sufficiently high values, then ADJUST-PLAN can increase a replanning window up to reach the earliest and latest time steps. This would determine a complete search.

Algorithm: ADJUST-PLAN
Input: a plan π_0 with some inconsistencies w.r.t. Π, a CPU-time limit `max-adjust-time`.
Output: either a correct plan or `fail`.

1. Let π be the plan obtained from π_0 by removing the actions that are not applicable for solving Π;
2. Identify the earliest time step `i` in π containing an inconsistency; if there is no such a time step, then return π;
3. If `i` is the latest time step of π, then set `init-time` to `i` $-$ 1 and `goal-time` to `i`, otherwise set `init-time` to `i` and `goal-time` to `i + 1`;
4. While CPU-time $\leq$ `max-adjust-time`
    5. Replan using as initial facts F(`init-time`) and as goals G(`goal-time`), where F(`init-time`) is the set of facts that are true at time `init-time`, and G(`goal-time`) is a set containing the preconditions of the actions in π at time `goal-time`;
    6. If there is no plan from F(`init-time`) to G(`goal-time`), or a search limit is exceeded, then decrease `init-time` or increase `goal-time` (i.e., we enlarge the replanning window), otherwise replace the existing actions between `init-time` and `goal-time` in π with the new subplan, and goto 2.
7. Return `fail`.

Figure 1: High-level description of ADJUST-PLAN.

Finally, during replanning within a particular window we impose a search limit that is automatically increased when the replanning window is enlarged.[2] The motivation of this heuristic is that when a replanning problem associated with a certain window is hard to solve, it can be easier to revise a larger portion of the plan (instead of dedicating a big effort to the revision of a restricted part of the plan). While this does not affect completeness, in practice it can be significantly effective for the efficient computation of an adapted plan.

2.2 A Detailed Description of ADJUST-PLAN

The input plan π_0 of ADJUST-PLAN is a plan for solving an old problem that we would like to adapt for solving a *new* problem Π, where the initial state or the goal state are different from the corresponding states in the old problem. We indicate with I_Π the initial state of Π, with π the plan under adaptation for solving Π, and with $\mathcal{G}$ the planning graph of Π. For the moment we assume that $\mathcal{G}$ is the planning graph for Π such that the last level of $\mathcal{G}$ contains all the goals of Π with no mutually exclusive relations between them. Moreover, we assume that $length(\mathcal{G}) = length(\pi_0)$, i.e., that

[2] In the current implementation this limit is defined by constraining either the maximum number m of time steps or the number of actions in the solution of the *re*planning problem, depending on the style of planning (either parallel or sequential) that we use for solving the replanning problem. For parallel planning, m is initially set to 3, and then it is automatically increased by 2 each time the replanning window is enlarged by one level; for sequential planning, m is initially set to 4, and then increased by 3 times the number actions in the replanning window.

the number of levels in $\mathcal{G}$ (excluding the final goal level) is the same as the number of time steps of π_0. In the last part of Section 2, we will generalise to the other cases.

The definition and computation of replanning window use the following notion of *applicable action*:

Definition 1 (Applicable action)**.** *An action A_k of π_0 at a certain time step k is* applicable *for solving Π if and only if A_k is a node of $\mathcal{G}$ at the action-level k.*

Note that, by definition of planning graph [1], when the fact corresponding to a precondition of an action A_k of π_0 is absent at the fact-level k of $\mathcal{G}$, also the action node corresponding to A_k is absent at the action-level k of $\mathcal{G}$.

Before starting the adaptation process we remove from π_0 the actions that are not applicable, and we set π to the resulting plan. During the adaptation process π is incrementally revised by replacing subplans within certain temporal windows with new subplans.

Definition 2 (Replanning window)**.** *Given two time steps i and j of a plan π under adaptation for solving Π, a* replanning window *for π is a pair $\langle F(i), G(j)\rangle$ such that:*

- *$0 \leq i < j$ and if $i > 1$, then the subplan of π from time step 0 to time step $i-1$ is a valid subplan for achieving the preconditions of the actions at time $i-1$ from I_Π;*
- *$F(i)$ is the set of positive facts that are true at time step i in π;*
- *$G(j)$ is a set of (sub)goals either consisting of the final goals of Π (if j is the time step corresponding to the final goals of Π), or containing the preconditions of the actions at time j (otherwise).*

The time steps i and j of a replanning window $\langle F(i), G(j)\rangle$ are called *replanning start time* and *replanning end time* respectively; $F(i)$ is called the *replanning initial state*, and $G(j)$ is called *replanning goal set*. Note that when the replanning goal state does not correspond to the goals of Π, Definition 2 requires that the replanning goal set $G(j)$ *contains* the action-preconditions at time j, but it does not specifies which are the other replanning goals that should be included (if any); moreover these extra goals are irrelevant for ensuring correctness and completeness [8]. On the other hand, as will be discussed in the next section, they can be useful for making the adaptation process more efficient.

Step 2 of ADJUST-PLAN guarantees that each replanning window contains the earliest inconsistencies of π. This allows computing an exact assessment of the facts $F(i)$ that are true at the replanning start time. (If the plan contained an inconsistency preceding the current replanning start time i, then we could compute only an approximation of the facts that are true at time i, because these can depend on how the preceding inconsistency will be repaired.) In particular, $F(i)$ can be computed in polynomial time by simulating the execution of the actions in π preceding time i, i.e., $F(i)$ can be incrementally computed according to the following definition:

$$F(i) = \begin{cases} I_\Pi & \text{if i} = 0 \\ E_i \cup F^+_{i-1} & \text{if i} > 1, \end{cases}$$

where E_i is the set formed by the positive effects of the actions of π at time step $i-1$, and F^+_{i-1} is the subset of $F(i-1)$ formed by the facts that are not deleted by the actions of π at time $i-1$.

In [8] we prove that ADJUST-PLAN is sound and complete, provided that `max-adjust-time` is set to a sufficiently high value and that a sound and complete algorithm is used for the replanning phase.

2.3 Heuristic Replanning Goals

According to Definition 2, the replanning goal set can contain some goals in addition to the preconditions of the actions that are planned at the replanning end time. In the following Σ_j indicates this set of action-preconditions, while Ω_j indicates the (possibly empty) set of the remaining goals of $G(j)$, i.e., $G(j) = \Sigma_j \cup \Omega_j$.

Including a non-empty Ω-set in $G(j)$ can be useful because some actions in the *subsequent* part of the plan might require that the Ω-facts hold in the replanning goal state (despite they are not preconditions of actions at time j). This need arises, for example, when an action A of π_0 in the replanning window has some effect ϕ that is not required by any other action in the replanning window, but that persists up to a time step after the replanning end time (j) to achieve a precondition of an action B, that otherwise would not be satisfied.[3] The inclusion of ϕ in $G(j)$ is useful because the (sub)plan found during replanning might not necessarily contain A (all the actions in the replanning window are always replaced by the new subplan solving the corresponding replanning problem). Thus, if ϕ were not in $G(j)$ and, after the subplan replacement, ϕ did not hold at time j, then ADJUST-PLAN would identify an inconsistency (the unsatisfied ϕ-precondition of B) at a time later than j, and hence another replanning phase would be activated.[4]

The Ω goal set of $G(j)$ can be seen as an assessment of the set Ω^*_j of the facts that should hold at time j for achieving those preconditions of actions in the subsequent part of π that otherwise would be unsatisfied (and so these actions would not be reusable for solving Π). Note that in general computing an exact assessment of Ω^*_j would not be feasible, because the subsequent part of π can change incrementally, as a consequence of repeated replanning phases to cope with inconsistencies at times later than j. Fortunately, including an exact assessment of Ω^* in $G(j)$ is not necessary for ensuring the correctness and completeness of the method [8]. Instead, the inclusion of the Ω-goals should be seen as a heuristic aimed at reusing as many actions of the original plan as possible, and at obtaining a fast adaptation.

We have developed definitions of Ω-goals, which are based on the assumption that Π can be solved by a plan which is similar to π, i.e., that the adaptation of π_0 requires a limited number of changes. In the rest of the paper, we present two of such definitions and we experimentally compare them in order to analyze their importance and effectiveness in practice.

[3] Using the causal-link planning terminology, we can say that π_0 contains the causal link $A \xrightarrow{\phi} B$.

[4] Reducing the amount of replanning can be crucial not only for efficiency, but also for reusing as much as possible the input plan, which is important, for example, in mixed-initiative planning.

We will use the following notation for associating a time step in the plan under adaptation π to a time step in the input plan π_0. Let t be a time step of π, $\hat{t}$ denotes the time step corresponding to t in π_0 before any modification. I.e., $\hat{t}$ is defined as $\hat{t} = t - \sum_{s=1}^{k} \delta_s$, where k is the number of subplan-replacements performed by the algorithm, and δ_i is the difference between the number of time steps involved in the new replanned subplan and the number of time steps in the corresponding existing subplan for the *i-th* replacement.

2.4 Forward Ω Goal set

We now give the definition of *forward* Ω *replanning goals* (Ω^F-goal set), which uses the notion of *persistent fact* with respect to π_0 and to the planning graph of the new problem. Ω_j^F is considered a set of facts that we include in $G(j)$ to obtain a replanning goal state that is similar to the corresponding state at time $\hat{\jmath}$ in π_0. The reason for this is that, if π_0 is correct for the old problem Π_0, then each state reached by π_0 contains a set of facts that is an exact assessment of its Ω^*-set. Hence, if we can solve Π using a plan that is similar to π_0, then we can expect that Ω_j is a good approximation of Ω_j^*.

Definition 3 (Persistent fact)**.** *A fact f_k is* persistent *in π_0 and $\mathcal{G}$ at time k if and only if f_k is a node of $\mathcal{G}$ at the fixpoint level such that the corresponding noop-action is* not *mutually exclusive with any applicable action of π_0.*

In the following definition $endtime(\pi_0)$ indicates the last time step of plan π_0.

Definition 4 (Forward Ω Goal set – Ω^F)**.** *Let k be a time step of π_0, i the earliest time step where π_0 has an inconsistency, j the end time of the current replanning window in π, and Γ_k the following set of facts:*

$$\Gamma_k = \begin{cases} F(k) & \text{if } k \leq i \\ E_k \cup \Gamma_{k-1}^{-} & \text{if } k > i, \end{cases}$$

where E_k is the set formed by the positive effects of the applicable actions of π_0 at time k, and Γ_{k-1}^{-} is the subset of Γ_{k-1} formed by the facts that are persistent in π_0 and $\mathcal{G}$ at time $k-1$. Ω_j^F is the subset of $\Gamma_{\hat{\jmath}}$ consisting of the facts that are persistent in π_0 and $\mathcal{G}$ at time $\hat{\jmath}$, if $\hat{\jmath} < endtime(\pi_0)$; the empty set if $\hat{\jmath} = endtime(\pi_0)$.

Note that the Γ-sets in the definition of the Ω^F-goals of $G(j)$, as well as the Σ-goals, can contain facts that are mutually exclusive in $\mathcal{G}$ at level $\hat{\jmath}$. This can be the case because the set E_j of action-effects, as well as the set Σ_j of action-preconditions, can contain facts that are mutually exclusive. This can happen even if the original plan from which we start the adaptation is valid (for the old problem), because the new problem Π can have a different initial state, introducing mutual exclusion relations that were not present in the planning graph of the old problem. During the computation of Ω_j^F, if a certain Γ_x contains a subset of mutually exclusive facts, then we impose that not more than one of them (randomly chosen) belongs to Γ_{x+1} (for $x < j$).

Figure 2 gives an example illustrating the elements of the sets $F(i)$, Γ_j, Ω_j^F and $G(j)$. The Figure shows a portion of the planning graph $\mathcal{G}$ for a problem Π in the

blocks world domain, that we are trying to solve by adapting a plan π_0 (for the sake of clarity the information in the planning graph that is not relevant for the example is omitted). Initially we have $\pi = \pi_0$. We assume that all the actions of π_0 are applicable, that the *earliest* inconsistency in π_0 is at time i and that the current replanning window is $\langle F(i), G(i+1)\rangle$. Moreover, we assume that, after the application of the actions of π preceding i, the set of positive facts in the replanning initial state is the same as Γ_i:[5]

$$\Gamma_i = F(i) = \{\texttt{hold\_C}, \texttt{on\_A\_B}, \texttt{on\_D\_E}, \texttt{clear\_D}, \texttt{clear\_A}\}.$$

Note that the facts `clear_B` and `ontab_D` are present in $\mathcal{G}$, but after the execution of π up to time $i-1$ they are false. `stack_C_B` is the only action of π at the time i. This action has two preconditions: `hold_C` and `clear_B`. `clear_B` is not satisfied at the level i, and this causes an inconsistency in π at the time step i. The elements of Γ_{i+1} are the union of:

- the set of the positive effects of the action `stack_C_B`: $\{\texttt{clear\_C}, \texttt{on\_C\_B}, \texttt{arm\_empty}\}$;
- the subset Γ_i^- of the facts in Γ_i that at time $i+1$ continue to be true (i.e. the facts in Γ_i that are persistent in π_0 and $\mathcal{G}$): $\{\texttt{clear\_D}, \texttt{on\_D\_E}, \texttt{clear\_A}\}$.[6]

`hold_C` does not belong to Γ_i^- (and to Γ_{i+1}), as the corresponding no-op is mutually exclusive with `stack_C_B` (i.e., `hold_C` is not persistent in π_0 and $\mathcal{G}$ at time i). Similarly, also `on_A_B` does not belong to Γ_{i+1}. The Σ-set and the Ω^F-set forming $G(j)$ are obtained as follows:[7] $\Sigma_{i+1} = \{\texttt{arm\_empty}, \texttt{clear\_D}, \texttt{ontab\_D}\}$, $\Omega^F_{i+1} = \{\texttt{on\_C\_B}, \texttt{clear\_C}, \texttt{clear\_A}\}$.

2.5 Causal Ω Goal set

The previously defined Ω^F-set contains all the facts that are persistent at a specific time step. Unfortunately, this set could contain some facts that are not necessary for the correctness of the remaining actions of the plan, which could make solving the local planning problem harder than necessary. If we consider the example of Figure 2, fact `clear_C` at time step $i+1$ is not necessary to the applicability of `pickup_D` and `stack_D_A`.

In order to improve the definition of the heuristic goal set, we can exploit the causal link representation of the plan under adaptation [16]: the existence of a causal link $a \xrightarrow{f} b$ "crossing" a time step l indicates that the truth of fact f at l is necessary for the correctness of π. More precisely, the set $\mathcal{L}^\pi$ of the causal links in π is defined as:

$$\mathcal{L}^\pi = \Big\{(a \xrightarrow{f} b) \mid a, b \in \pi \ \wedge \ f \in add(a) \ \wedge \ f \in prec(b) \ \wedge \ Lev_\pi(a) < Lev_\pi(b) \ \wedge$$
$$\nexists c \in \pi \ \texttt{s.t.} \ f \in del(c) \ \wedge \ Lev_\pi(a) \leq Lev_\pi(c) < Lev_\pi(b)\}\,,$$

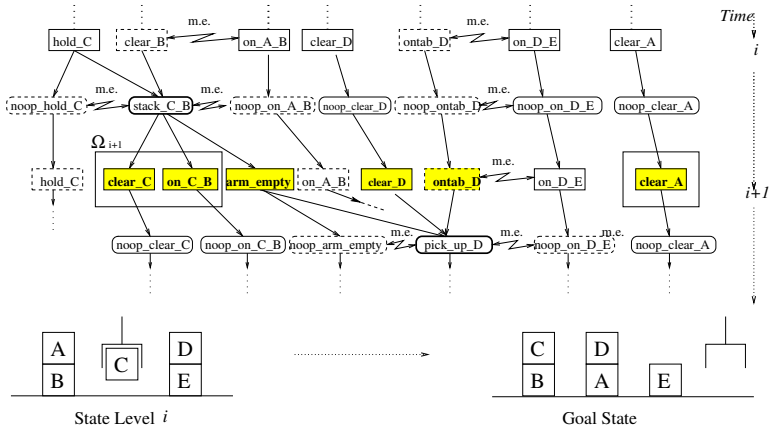
where $Lev_\pi(x)$ corresponds to the time step of the action x in π. Moreover, we define the set $\mathcal{L}^\pi_l \subseteq \mathcal{L}^\pi$ of the causal links in π crossing a time step l as follows:

$$\mathcal{L}^\pi_l = \Big\{(a \xrightarrow{f} b) \in \mathcal{L}^\pi \mid Lev_\pi(a) < l \leq Lev_\pi(b)\Big\}\,.$$

[5] Since in this example $\hat{k} = k$ for any step k, to simplify the notation $\hat{k}$ is indicated with k.

[6] `on_D_E`, `clear_D` and `clear_A` are the only ones in $F(i)$ whose corresponding no-ops at the level i in the planning graph are not mutually exclusive either with `stack_C_B` or with each other.

[7] `arm_empty` and `on_D_E` in Γ_{i+1} are not in Ω^F_{i+1} because they are not persistent in π and $\mathcal{G}$ at $i+1$.



Goals={on_C_B, on_D_A}, $\pi = \{...; i : \texttt{stack\_C\_B};\ i+1 : \texttt{pickup\_D};\ i+2 : \texttt{stack\_D\_A}\}$.

Figure 2: Example illustrating the definitions of $F(i)$, Γ_j, Ω_j^F and $G(j)$, for $j = i+1$. The actions of π at the times i and $i+1$ are stack_C_B and pickup_D respectively. We use solid boxes for representing the facts belonging to $F(i)$, and dashed boxes for the facts of $\mathcal{G}$ that are not in $F(i)$. The facts belonging to $G(j)$ are indicated with gray boxes, while the facts belonging to Ω_j^F are enclosed into a box.

The f-facts involved in the causal links of $\mathcal{L}_l^\pi$ can be used to derive a new definition of the Ω-goal set at time step l:

$$\Omega_l^C = \left\{ f \mid \exists (a \xrightarrow{f} b) \text{ in } \mathcal{L}_l^\pi \right\}.$$

For the example of figure 2, we have the following causal links crossing time step $i+1$ ($\texttt{a}_{\texttt{init}}$ and $\texttt{a}_{\texttt{end}}$ are special actions with effects the facts that are true in the initial state, and with preconditions the problem goals, respectively):

$$\mathcal{L}_{i+1}^\pi = \left\{ \left(\texttt{stack\_C\_B} \xrightarrow{\texttt{on\_C\_B}} \texttt{a}_{\texttt{end}}\right), \left(\texttt{stack\_C\_B} \xrightarrow{\texttt{arm\_empty}} \texttt{pickup\_D}\right), \right.$$
$$\left. \left(\texttt{a}_{\texttt{init}} \xrightarrow{\texttt{clear\_D}} \texttt{pickup\_D}\right), \left(\texttt{a}_{\texttt{init}} \xrightarrow{\texttt{clear\_A}} \texttt{stack\_D\_A}\right) \right\}$$

and so we have $\Omega_{i+1}^C = \{\texttt{on\_C\_B}, \texttt{clear\_A}, \texttt{arm\_empty}, \texttt{clear\_D}\}$.

Differently from Ω_{i+1}^F, Ω_{i+1}^C does not contain fact clear_C, but it includes clear_A, which is necessary to the applicability of stack_D_A (note that facts arm_empty and clear_D already belong to Σ_{i+1}). It is important to observe that, since π could be an invalid plan, there could be some conflicts between facts in Ω_l^C and in Σ. In these cases, in order to avoid having conflicting replanning goals, it is necessary to select a subset of the computed replanning goals. We do this heuristically. Intuitively, having to choose between two facts that are mutually exclusive, we prefer the fact that (a) is used by an action that is temporally nearest to the time step of the replanning goals and (b) whose truth satisfies the greatest number of action preconditions.

2.6 Additional Details

To conclude the description of our plan-adaptation method, we consider the cases in which the number of time steps n_t involved in π_0 is different from the number of levels

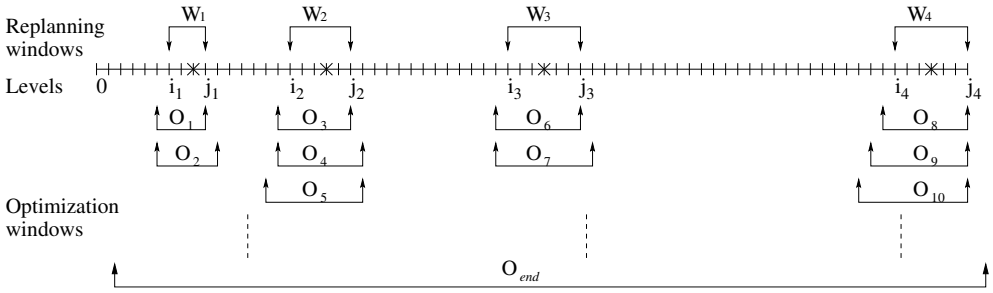


Figure 3: Example illustrating the iterative optimization procedure. $W_{1...4}$ are the replanning windows of the subplans replaced during adaptation. O_i are the alternative replanning windows considered during the optimization process so as to find a series of plans each of which is characterized by a smaller number of time steps/actions with respect to the previous plan.

n_l in the planning graph $\mathcal{G}$.

When $n_t > n_l$, the applicable actions of π_0 are determined by first extending $\mathcal{G}$ to have the same number of levels as the time steps of π_0, and then applying a definition of an applicable action analogous to Definition 1.

When $n_t < n_l$, the mapping between plan-actions and graph-nodes in the definition of applicable actions is done by considering the last n_t levels of $\mathcal{G}$. E.g., we try to map actions at time n_t to nodes at the last level of $\mathcal{G}$, actions at time $n_t - 1$ to nodes at the penultimate level of $\mathcal{G}$, and so on. If for an action A of π_0 there is no corresponding action node in $\mathcal{G}$, then we consider A an action that cannot be applied for solving Π, and we remove it from π. If $n_t < n_l$, we say that a fact f_k is *persistent* in π_0 and $\mathcal{G}$ if and only if f_k is a node of $\mathcal{G}$ at level $l = k + length(\mathcal{G}) - length(\pi_0)$, and the corresponding noop-action at level l is not mutually exclusive with any applicable action of π_0 at time k.

2.7 Plan Optimisation

Concerning plan quality, clearly, the plan adaptation techniques that we have presented can find a first solution that is not optimal with respect to the number of actions or time steps. If we are interested in finding good quality plans, which is important in many applications, we can extended our method by identifying a "suitable" alternative replanning window O, for each previous replanning phase, and repeating the adaptation process in order to find a new subplan with a lower number of actions/time steps w.r.t. the corresponding number of actions/time steps in O.

The iterative optimization process illustrated in the example of Figure 3 produces a series of optimized plans, each of which involves a smaller number of either time steps or actions. The longer this process is, the better the final plan can be. This process can be interrupted at any time to return the best plan generated so far.

The computation of an optimized plan is based on considering alternative replanning windows for the subplans that were previously inserted, during either the plan adaptation phase or the plan optimization process. For each of these subplans, we have to consider a replanning window that is larger than the one previously used; if,

for this enlarged window, we find a subplan involving a smaller number of time steps (or actions) w.r.t. the current subplan in the replanning window under consideration, then the plan actions inside the replanning window are replaced with those in the newly generated subplan.

For instance, if we have replanned from $F(\hat{\imath})$ to $G(\hat{\jmath})$, obtaining a subplan with k time steps, then we will optimize from $F(i')$ to $G(j')$ with $j' > \hat{\jmath}$ and $i' \leq \hat{\imath}$, imposing to the solution subplan a maximum number of time steps equal to $(j' - \hat{\jmath}) + k + (\hat{\imath} - i')$. If there is no plan from $F(i')$ to $G(j')$ that satisfies such a constraint, then we either decrease i' or increase j' (i.e. we enlarge the replanning window); otherwise, we revise π_0 with the new subplan, and then we consider another replanning window previously processed by the the plan adaptation/optimization process for the possible improvement of the corresponding subplan. In principle, each replanning window can be enlarged up to contain the entire current plan, i.e., $F(i') = I_{\Pi_0}$ (the initial state) and $G(j') = G_{\Pi_0}$. In this case, if we solve the replanning problems using an optimal planner, like IPP, obviously the generated plan is guaranteed to be optimal.

3 Experimental Results

We present the results for some experiments aimed at testing the efficiency of our plan adaptation approach, which is implemented in the ADJ system. The general goal of our experiments was to confirm the hypothesis that plan adaptation using ADJ can be much faster than replanning from scratch, for which we used two types of planners: the optimal parallel planner IPP based on planning graphs, and the satisficying state-based sequential planner FF based on heuristic search techniques. These well-known planners are among the best available for these styles of planning. We tested our system using a large collection of known planning problems in different known domains: Rocket [20], Logistics [12], Gripper [4], DriverLog and Zenotravel [14]. For each of these problems we designed some modifications (to the initial state or to the goals) and we solved the modified versions using a solution plan for the original problems as part of the input.[8]

Concerning the comparison of ADJ with IPP, we used a set of variants of some benchmarks from the 1st International Planning Competition (IPC). For this experiment, IPP was also used by ADJ to solve the planning problems associated with the replanning windows. Figure 4 compares the CPU-times consumed by ADJ with the CPU-times consumed by IPP for solving the same set of problem variants, which are named on the x-axis of the plots using numbers. Each problem variant contains few changes to the facts of the the initial or goal state(s) of the original problem, making the input plan solving the original problem invalid for the revised problem.[9] In general, these results show that solving a problem by plan adaptation using ADJ can be much faster than by replanning from scratch, up to four orders of magnitude. (For example, ADJ with Ω^C solved the Logistics_c problem 14 in 0.66 seconds and with Ω^F in 0.75 seconds; while IPP required 1871 seconds.) The main reason of this observed behaviour

[8] The machine we used for the tests was an Intel(R) Xeon(TM) CPU 2.40GHz, with 1GB of RAM.

[9] The formalisation of the test problems is available from
`http://www.ing.unibs.it/~serina/adaptation/adjust-problems.tar`

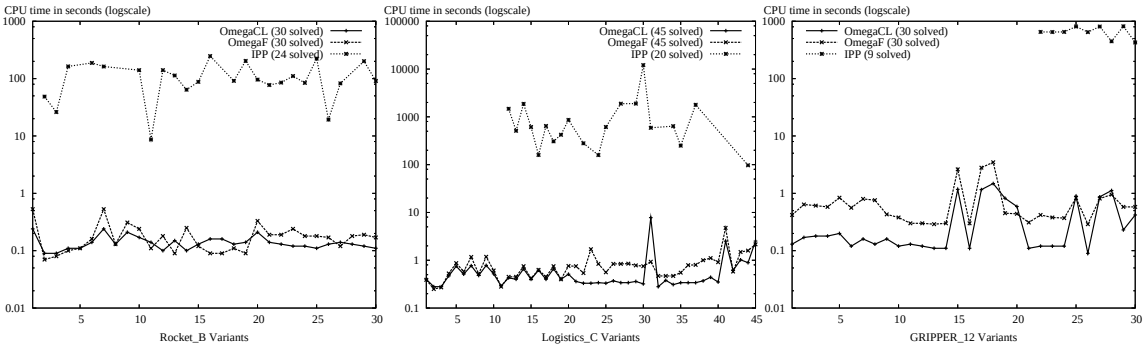


Figure 4: CPU-seconds (logarithmic scale) required by ADJ and IPP for solving variants of Rocket_b, Logistics_c, and Gripper_12.

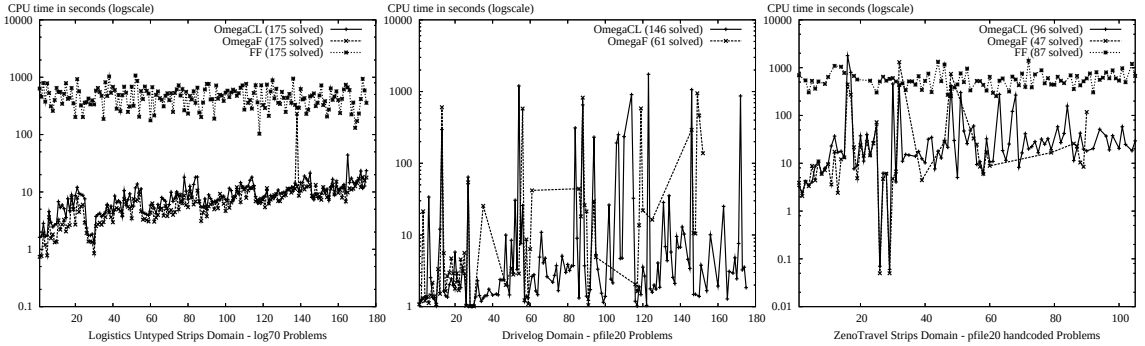


Figure 5: CPU-seconds (logarithmic scale) required by ADJ and corresponding generation CPU-time using FF for the variants of Logistics70, DriverLog Strips pfile20 and Zenotravel Strips Handcoded pfile20.

is that, very often, the planning subproblems defined by the replanning windows are much easier than the complete full problem considered by IPP, and they can be solved by shorter (sub)plans. In general, the performance of ADJ depends on the number of the replanning windows considered, on their size, and on the complexity of the corresponding replanning problems. But we observed that the first two factors seem less crucial than the third one.

From the results in Figure 4 we can also observe that, for this experiment, the performance of ADJ with Ω^C and with Ω^F are comparable. We believe that the main reason here is the simplicity of the test problems, for which the use Ω^F is sufficiently effective. On the other hand, we observe that the overhead due to the computation of casual links in Ω^C does not slow down the overall performance.

The problems used for the experimental analysis presented in the next part of this section are variants of hard problems from the 2nd and 3rd IPCs. None of these problems can be solved by IPP within the CPU-time and memory limits imposed in our experiments. Since FF was the winner of the 2nd IPC (domain-independent track), for these problems we used FF both as the reference planner for the replanning from scratch and as the planner for solving the local replanning problems in ADJ. For this experiment, the mutual exclusive relations used by ADJ were computed by running the

DISCOPLAN system [6]. Figure 5-a shows the results for several variants of the very hard problem logistics-70 (from the "additional problems track" of the 2nd IPC). We can observe that plan adaptation is about two orders of magnitude faster than planning from scratch. The performances of Ω^C and Ω^F are again very similar; sometimes the less accurate Ω^F heuristic is even more effective than the Ω^C heuristic, but we observed that often the solutions generated using Ω^F are not as good as to those obtained using Ω^C.

Figure 5-b gives results for the Driverlog domain from the 3rd IPC [14]. We created a set of variants to the pfile20 problem (the most difficult STRIPS problem of the "standard track" of the 3rd IPC). These modifications concern the initial and/or final positions of the involved drivers and packages. FF solved none of these problem variants within the 3600 CPU-seconds limit. Differently from the previous cases, in this experiment we observed that the Ω^C goal sets are very useful compared to the Ω^F goal sets: ADJ with Ω^C solves 146 over 175 problem variants, while ADJ with Ω^F solves only 61 variants. Moreover, ADJ with Ω^C is generally faster than with Ω^F, and it finds solutions with better quality. Interesting, Ω^F can solve two problems that Ω^C cannot solve. We think that the reason is related to the fact that ADJ with Ω^C can remove some useless actions from the plan, which allows the system to find better quality solutions, but which for these problems makes processing the final replanning windows more difficult.

Figure 5-c gives results for a set of variants of the pfile20 "handcoded" problem in the Zenotravel STRIPS domain (the hardest problem in this domain, originally designed for testing the performance of domain-*de*pendent planners). Here again we can observe a generally good behavior of Ω^C, with which ADJ solves 96 over 105 problem variants, while with Ω^F it solves only 47 variants. FF solves 87 variants (using at most 1800 CPU-seconds). Overall, ADJ with Ω^C is one or two orders of magnitude faster than FF, although the first solution produced by ADJ can contain more actions than the corresponding solution produced by FF.

As previously observed, the first plan computed by ADJ can involve a number of time steps or actions that is higher than necessary. In Figure 6, we give some experimental results about the the effectiveness of the plan optimisation phase of ADJ. On the x-axis, we have the CPU-seconds for finding a solution; on the y-axis, we have plan qualities, in terms of either number of time steps (levels) or number of actions, for the various solutions that are incrementally generated during the optimisation process.

The curves in Figure 6 indicate that ADJ computes very quickly a first solution which is worse than the optimal solution. However, then the optimisation process can incrementally derive additional solutions with better quality, up to a solution that in terms of number of involved time steps or actions is similar to the optimal one. For example, concerning the Logistics_a variant 24, ADJ computes a first solution involving 16 time steps in 1.24 seconds. Then the optimization process derives additional solutions involving 15, 14, 13, 12, and 11 time steps, using 1.8, 4.2, 142, 1599, 1800 CPU-seconds, respectively. While IPP computes an optimal solution in 1430 CPU-seconds.

Moreover, Figure 6 also compare the performances of FF and ADJ with either Ω^C or Ω^F. Although usually the first solution computed by ADJ is not better than the one generated by FF, we observe that, for the test problems considered in this

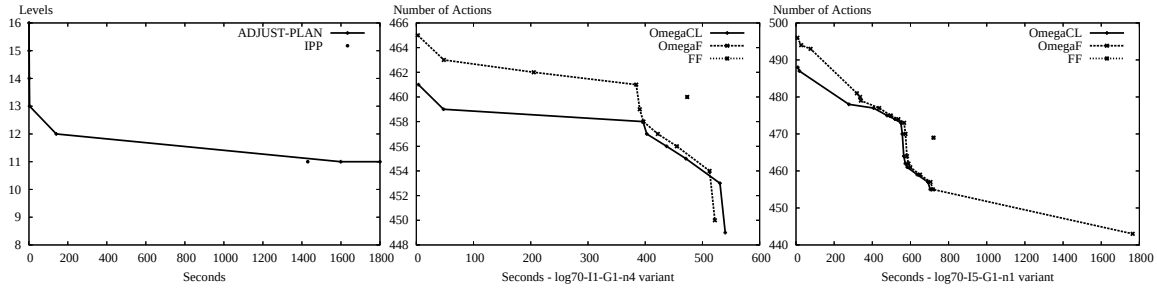


Figure 6: Optimisation phase for variant num. 24 of Logistics_a considering ADJ vs. IPP and variants 34 and 151 of Logistics70 Untyped considering ADJ vs. FF

experiment, ADJ generates additional plans quickly deriving a solution that is better than the solution computed by FF.

4 Conclusions and Future Work

We have presented a method and some heuristics for efficient domain-independent plan adaptation that are implemented in the ADJ system. An experimental analysis indicates that solving a planning problem by adapting an existing plan for a similar problem using ADJ can be dramatically more efficient than replanning from scratch using two well-known approaches of planning.

In the literature, other methods and systems for plan adaptation have been proposed, such as PRIAR [11], PRODIGY/ANALOGY [20] and SPA [9]. These systems use underlying planning algorithm and data structures that are significantly different from those implemented in ADJ. Another important difference concerns the input information. While ADJ uses a very simple and general plan description that can be easily generated from the output of many planners (or even written by hand), the other mentioned systems use specific additional information (such as the "refinement" decisions taken during the search process in SPA) constraining the input plan to be a plan generated by a specific planner. Other recent related systems are FAR-OFF [17] and van der Krogt & de Weerdt's system [19], each of which is based on techniques that are significantly different from ADJ's techniques.

We are currently studying a new version of ADJ using weaker, but computationally more efficient data structures and an extension to support numeric and temporal domains [14]. Other current and future work includes further techniques for determining replanning goals, and additional experiments, including, in particular, a comparison with the performance of other related plan adaptation systems. (A preliminary comparison with the two most related systems mentioned above indicates that ADJ performs more efficiently.) Moreover, we are studying the integration of LPG [5] and SGPLAN [21] into ADJ for solving the problems associated with ADJ's replanning windows.

Finally, we intend to develop a complete case-based planning system, which selects an existing plan from a library and adapts it to solve the current problem using ADJ. Since adapting an existing plan is PSPACE-Complete [15], the use of an exact general method for deciding whether adapting a plan from a library gives a computational advantage w.r.t. a complete replanning from scratch seems practically infeasible. However, this question can be addressed by using heuristic methods such as the polynomial

technique presented in [17].

References

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

[2] B. Drabble. Excalibur — A program for planning and reasoning with processes. *Artificial Intelligence*, 62(1):1–40, 1993.

[3] G. Ferguson and J. Allen. Towards a mixed-initiative planning assistant. In *In Proceedings of AIPS-96*, 1996.

[4] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *Proceedings JCAI-99* , pages 956–961, 1999.

[5] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)*, 20:pp. 239–290, 2003.

[6] A. Gerevini and L. Schubert. Inferring state constraints in DISCOPLAN: Some new results. In *Proceedings of the AAAI-00*, pages 761–767. AAAI press/The MIT Press, 2000.

[7] A. Gerevini and I. Serina. Fast planning through greedy action graphs. In *Proceedings of the AAAI-99*, pages 503–510. AAAI Press/MIT Press, July 1999.

[8] A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the AIPS-00*, pages 112–121. AAAI Press/MIT Press, 2000.

[9] S. Hanks and D.S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research (JAIR)*, 2:319–360, 1995.

[10] J. Hoffmann. FF: The fast-forward planning system. *AI Magazine*, 22(3):57–62, 2001.

[11] Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.

[12] H.A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In Howard Shrobe and Ted Senator, editors, *Proceedings of the AAAI-96*, pages 1194–1201. AAAI Press, 1996.

[13] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Fourth European Conference on Planning (ECP'97)*, pages 273–285. Springer Verlag, 1997.

[14] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research (JAIR)*, Forthcoming, 2003.

[15] B. Nebel and J. Koehler. Plan reuse versus plan generation: A complexity-theoretic perspective. *Artificial Intelligence*, 76:427–454, 1995.

[16] J.S. Penberthy and D.S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR'92*, pages 103–114, Boston, MA, 1992. Morgan Kaufmann.

[17] Flavio Tonidandel and Marcio Rillo. The far-off system: A heuristic search case-based planning. In *Proceedings of AIPS*, pages 302–311. AAAI Press, 2002.

[18] D.R. Traum, J.F. Allen, G. Ferguson, P.A. Heeman, Chung-Hee Hwang, T- Kato, N. Martin, M. Poesio, and L. K. Schubert. Integrating natural language understanding and plan reasoning in the TRAINS-93 conversation system. In *Working Notes of the AAAI Spring Symposium on Active NLP*, pages 63–67, Stanford, CA, 1994.

[19] R. van der Krogt and M. de Weerdt. Plan repair as an extension of planning. In *Proceedings of ICAPS'05*, pages 161–170. AAAI, 2005.

[20] M. Veloso. *Planning and learning by analogical reasoning*, volume 886 of *Lecture Notes in Artificial Intelligence and Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1994.

[21] Hsu C. W., Wah B. W., Huang R., and Chen Y. X. New features in sgplan for handling soft constraints and goal preferences in PDDL3.0. In *Abstract Booklet of the Fifth International Planning Competition,ICAPS'06*, 2006.