Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques

Alfonso Gerevini and Ivan Serina
Dipartimento di Elettronica per l'Automazione
Università di Brescia, via Branze 38, 25123 Brescia, Italy
{gerevini,serina}@ing.unibs.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 propose a domain-independent method for plan adaptation that combines two techniques. The first technique 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 is solved using systematic search for Planning Graphs. The second technique modifies the original plan using local search for Action Graphs, which are particular subgraphs of a planning graph. This technique can be used either for solving a plan adaptation task or as a preprocessing for reducing the number of inconsistencies in the input plan. Experimental results show that in practice adapting a plan using our techniques can be very efficient.

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 (Ferguson and Allen 1994; 1996), 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 (Nebel and Koehler 1995). 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.

GPG is a domain-independent planning system based on planning graphs (Blum and Furst 1995), that uses a collection of local and systematic search techniques for solving both plan generation and plan adaptation tasks.[1] In this context a plan is 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). In (Gerevini and Serina 1999) we introduced GPG focusing on plan generation. In this work we focus on plan-adaptation, presenting an efficient method based on two techniques using different search algorithms.

The first technique 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 (Blum and Furst 1995). 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 search algorithm (in our implementation we used the search algorithm of IPP (Koehler *et al.* 1997), 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 new problem is solvable, then the revision of the input plan for the old problem leads to a correct plan for the new problem.

The second plan-adaptation technique modifies the original plan by performing local search for planning graphs (Gerevini and Serina 1999a; 1999). In this framework the problem of adapting an input plan is

[1] GPG is available by inquiring the authors.

formulated as a search problem where the elements of the search space are particular subgraphs of a planning graph. Each of these subgraphs represents a partial plan under adaptation, that can contain some inconsistencies. The initial subgraph is constructed using the planning graph for the new problem and the plan for the old problem, while a final subgraph is a graph representing a valid plan for the new problem. Typical operators for moving from a search state to the next one are some modifications of the current subgraph corresponding to adding (or deleting) certain actions to (from) the partial plan represented by the subgraph.

In the rest of the paper, first we give a general description of the technique using systematic search, followed by a more detailed, formal description; then we describe the technique using local search and its combination with the systematic technique; finally, we present experimental results showing that in practice our approach can be very efficient, and we give our conclusions.

Systematic search for Plan Adaptation

In this section we describe a plan adaptation technique using systematic search. We assume that the reader is familiar with the Graphplan-style of planning (Blum and Furst 1995).

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 systematic plan adjustment process of GPG 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 next sections.

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 (the proof is given in the next section). Figure 1 describes the main steps ADJUST-PLAN, while a more formal description is given in the next section.[2] 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 currently considering STRIPS 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).

Algorithm: ADJUST-PLAN
Input: a plan π_0 containing some inconsistencies with respect to Π, and 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. Systematically 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.

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` using systematic search. 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`; see Figure 2). 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`).[3]

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 systematic replanning at step 5 is performed by first constructing the corresponding

[2] A general, less efficient version of ADJUST-PLAN is also sketched in (Gerevini and Serina 1999).

[3] GPG includes both systematic and local search techniques that can be used in combination. When `max-adjust-time` is exceeded, GPG can activate the local search phase, as described in the second part of the paper.

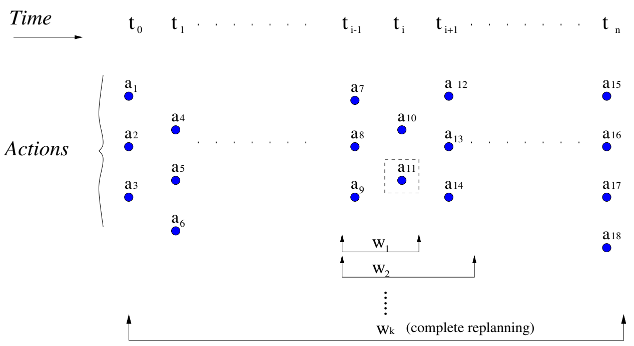

Figure 2: Example of replanning windows for a plan in which the earliest inconsistency is at time t_i and consists of an unsatisfied precondition of action a_{11}. W_1 is the initial replanning window, and can be incrementally enlarged up to include all the actions of the plan.

*re*planning graph, and then searching it by using the same backtracking scheme as IPP (Koehler *et al.* 1997). This search method guarantees that if a (sub)plan is found, then this is optimal with respect to the number of time steps that are involved.

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.[4] 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). GPG 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.

Finally, during replanning within a particular window we impose a search limit that is automatically increased when the replanning window is enlarged.[5] 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.

A Detailed Description of ADJUST-PLAN

This section and the next one are devoted to a detailed, formal description of ADJUST-PLAN. The input plan π_0 of ADJUST-PLAN is either:

- 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; or
- a modification of an old plan that we have revised through local search for reducing the number of inconsistencies before running ADJUST-PLAN (the use of local search as preprocessing will be presented in the next section).

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 with the minimum number of levels such that the last level of $\mathcal{G}$ contains all the goals of Π with no mutually exclusive relations between them.[6] Moreover, we assume that $length(\mathcal{G}) = length(\pi_0)$, i.e., that the number of levels in $\mathcal{G}$ (excluding the final goal level) is the same as the number of time steps of π_0. At the end of the next section we will generalize 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 (Blum and Furst 1995), 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:*

- *$0 \le i < j$ 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*

[4] When the replanning window is enlarged by moving the goal state forward, keeping the same initial state, we use the memoization of unachieved subgoals to prune the search, as indicated in (Blum and Furst 1995; Koehler *et al.* 1997).

[5] In the current implementation this limit is defined by constraining the maximum number of levels in the *re*planning graph, that is initially set to 3, and then automatically increased by 2 each time the replanning window is enlarged by 1 time step.

[6] This is the same graph that would be initially constructed by IPP for solving Π. The construction of such a graph allows us to detect some cases in which Π is unsolvable (Blum and Furst 1995; Koehler *et al.* 1997).

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). As will be clear from the proof of the theorem stating correctness and completeness of ADJUST-PLAN, these extra goals are irrelevant for ensuring correctness and completeness. 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 starts 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 $i-1$.

The following theorem states that ADJUST-PLAN is sound and complete. The proof uses 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. For example, suppose that $k = 1$ and that the algorithm replans from $F(2)$ to $G(5)$, replacing a subplan involving 3 time steps with a new subplan involving 5 time step. We will have $\delta_1 = +2$, and $\hat{t} = 7$ when t = 9.

Theorem 1 ADJUST-PLAN *is sound and complete (provided that* `max-adjust-time` *is set to a sufficiently high value).*

Proof. *Soundness.* Suppose that the output plan π_a is an adaptation of the input plan that is not correct for Π. This implies that (a) one of the goals of Π is not achieved by π_a, or (b) π_a contains an action precondition which is not satisfied, or (c) π_a contains a pair of unordered actions that are planned at the same time and interfere or produce inconsistent effects (i.e, using the planning graph terminology, π_a contains two mutually exclusive actions). We show that none of these conditions can hold at the end of the adaptation phase. We have that:

(1) by step 2 of ADJUST-PLAN each time step where the plan contains some inconsistency that is identified and then processed is the *earliest* time step containing an inconsistency in the current plan.

(2) Since the replanning search algorithm (based on IPP) is sound (Koehler *et al.* 1997), each new replanned subplan does neither involve mutually exclusive actions, nor actions containing unsatisfied preconditions.

(3) After having replaced a subplan from some time step i to some other time step j ($i < j$) in step 6, the algorithm repeats step 2, identifying the time step to be processed (if any). This is done by simulating the execution of the plan starting from $F(i)$ and using the actions of the new subplan, up to time step h, followed by the applicable actions of the input plan occurring after time step $\hat{j}$ (if any), where h is the number of time steps involved by the new subplan.

(4) The plan-execution simulation (forward temporal projection) performed at step 2 computes for each time t that is reached, all the facts that are true at time t.[7]

From (1)—(4) and Definition 2 (replanning window) it follows that each time the algorithm replaces at step 6 a subplan for achieving $G(q)$ from $F(p)$, for some p and q, the subplan in the resulting plan from $F(0)$ (the initial state of Π) to $G(q)$ is a valid plan for achieving $G(q)$ from $F(0)$.

Let $G(l)$ be the last replanning goal set considered by the algorithm. If $G(l)$ is the goal set of Π, then, by the previous considerations and Definition 2, the algorithm computes a valid plan from $F(0)$ to the goals of Π. Hence, none of (a), (b), and (c) can hold. If $G(l)$ is not the goal set of Π, then, by the definition of $G(l)$ and the previous considerations, the subplan from $F(0)$ to $G(l)$ is a valid subplan for achieving the preconditions of the actions at time step l. Moreover, since l is the last time step identified by the algorithm where π plan contains an inconsistency, the subplan of π from l to the last time step of π is a valid plan for achieving the goals of Π from $F(l)$. Hence, also in this case none of (a), (b) and (c) can hold.

Completeness. If the new problem (Π) is solvable, then ADJUST-PLAN finds a plan, otherwise it returns `fail`. This is a straightforward consequence of the fact that the size of the replanning window for fixing the inconsistencies identified at a certain time step is monotonically increased in step 6 until either

- a valid subplan is found within a certain search-limit (e.g., a maximum number of time steps allowed in the planning graph of the subplan), or
- the initial and last time steps of π are reached.

[7]Since we are assuming the closed world assumption, it also implicitly computes all the facts that are false.

In the second case the replanning window has reached its maximum size, and is then kept constant, while the search limit is removed and the size of the planning graph is increased of one level whenever the (systematic) search fails. This determines the same complete search as in the standard search scheme of IPP applied to Π (Koehler *et al.* 1997). □

Heuristic Replanning Goals

According to Definition 2, the replanning goal set may 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.[8] 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.[9]

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, from the proof of the previous theorem it is clear that including an exact assessment of Ω^* in $G(j)$ is not necessary for ensuring the correctness and completeness of the method. 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.

In our current implementation we have considered a relatively simple definition of Ω-goals, which is based on the assumption that Π can be solved by a plan similar to π, i.e., that the adaptation of π_0 requires a limited number of changes. Ω_j is considered as 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 $\hat{j}$ in π_0. The motivation of 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 similar to π_0, then probably Ω_j is a good approximation of Ω_j^*. If π_0 is a modification of a plan for Π_0 that we have obtained through local search for reducing the number of inconsistencies, then π_0 might be incorrect for Π_0. However, in this case π_0 is "almost" correct for the *new* problem Π, and thus Ω_j can be considered as an assessment of Ω_j^* for the new problem.

We now give a formal definition of Ω-goals which uses the notion of *persistent fact* with respect to π_0 and to planning graph of the new problem. We indicate with $endtime(\pi_0)$ the last time step of π_0.

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 level k such that the corresponding noop-action at level k is* not *mutually exclusive with any applicable action of π_0 at time k.*

Definition 4 (Ω Goal set) *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:*

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

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 is the subset of $\Gamma_{\hat{j}}$ consisting of the facts that are persistent in π_0 and $\mathcal{G}$ at time $\hat{j}$, if $\hat{j} < endtime(\pi_0)$; the empty set if $\hat{j} = endtime(\pi_0)$.

Note that the Γ-sets in the definition of the Ω-goals of $G(j)$, as well as the Σ-goals, can contain facts that are mutually exclusive in $\mathcal{G}$ at level $\hat{j}$. 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, if a certain Γ_x contains a subset of mutually exclusive facts, then we impose that at most one of them (randomly chosen) belong to Γ_{x+1} (for $x < j$). Finally, note that for any j, Ω_j is independent from the modifications performed to π by ADJUST-PLAN, and can be computed before running this algorithm.

Figure 3 gives an example illustrating the elements of the sets $F(i)$, Γ_j, Ω_j and $G(j)$. The Figure shows a portion of the planning graph $\mathcal{G}$ for a problem Π in the blocks world, that we are trying to solve by adapting a plan π_0 (for the sake of clarity the information in the

[8] In terms of the causal-link planning terminology, we have that π_0 contains the causal-link $A \xrightarrow{\phi} B$.

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

planning graph that are not relevant for the example are 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 execution simulation of the actions of π preceding i, the set of positive facts in the replanning initial state is the same as Γ_i:[10]

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

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 time i. This action has two preconditions: hold_C and clear_B. clear_B is not satisfied at level i, and this causes an inconsistency in π at time 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 $\mathcal{G}$): $\{\texttt{clear\_D}, \texttt{on\_D\_E}\}$.[11]

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-set forming $G(j)$ are the following sets:[12]

$$\Sigma_{i+1} = \{\texttt{arm\_empty}, \texttt{clear\_D}, \texttt{ontab\_D}\},$$
$$\Omega_{i+1} = \{\texttt{on\_C\_B}, \texttt{clear\_C}\}.$$

When the inconsistencies of π_0 are all at the same time step, or when during the adaptation process we have reached the last time step where π contains an inconsistency, it is possible to compute an exact assessment of Ω_j^*. This can be done by a backward temporal projection of the goals ($goals(\Pi)$) of Π through the actions of π.

Definition 5 (Backward Ω-goal set) *Let $\langle i, j\rangle$ be a replanning window such that π does not contain inconsistencies after j, and Θ_x the following set of facts:*

$$\Theta_x = \begin{cases} goals(\Pi) & \text{if } x = endtime(\pi) \\ \Sigma_x \cup \Theta_{x+1}^- & \text{otherwise,} \end{cases}$$

where $x \geq j$, Σ_x is the set of positive preconditions of the actions in π at time x, and Θ_{x+1}^- is the subset of Θ_{x+1} formed by the facts that are not achieved by any action of π at time x. The backward Ω-goal set *Ω_j^b is defined as: $\Omega_j^b = \Theta_j - \Sigma_j$.*

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

[11]In facts we have that on_D_E and clear_D are the only facts of $F(i)$ whose corresponding no-ops at level i in the planning graph are mutually exclusive neither with stack_C_B nor with each other.

[12]The facts arm_empty and on_D_E of Γ_{i+1} do not belong to Ω_{i+1}, because they are not persistent in π and $\mathcal{G}$ at time $i+1$, i.e., the corresponding no-ops are not mutually exclusive with pickup_D, which is an action of π at level $i+1$.

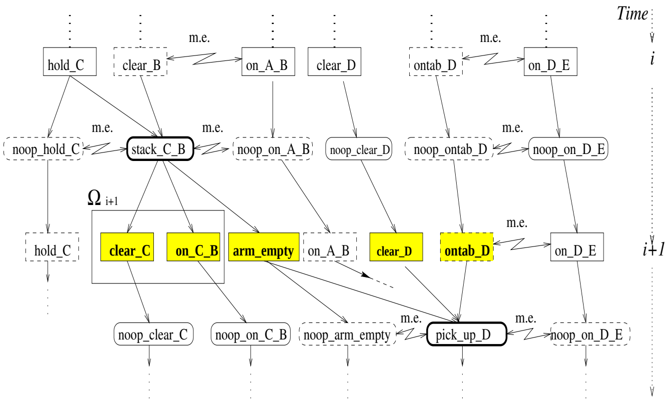

Figure 3: Example illustrating the definitions of $F(i)$, Γ_j, Ω_j and $G(j)$, for $j = i+1$. The actions of π at time i and $i+1$ are stack_C_B and pickup_D respectively. We use solid boxes for representing facts belonging to $F(i)$, and dashed boxes for facts of $\mathcal{G}$ that are not in $F(i)$. The facts belonging to $G(j)$ are indicated with gray boxes, while the facts belonging to Ω_j are enclosed into a box.

To conclude the description of our systematic plan-adaptation method, we consider the cases in which: (a) the number of time steps involved in π_0 is higher than the number of levels in $\mathcal{G}$, (b) the number of the time steps involved in π_0 is smaller than the number of levels in $\mathcal{G}$. In case (a) 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 applicable action analogous to Definition 1. In case (b), if π_0 involves n time steps, then the mapping between plan-actions and graph-nodes in the definition of applicable actions is done by considering the last n levels of $\mathcal{G}$. E.g., we try to map actions at time n to nodes at the last level of $\mathcal{G}$, actions at time $n-1$ 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. In case (b) 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.

Theorem 1 can be easily extended to the cases where $length(\pi_0) \neq length(\mathcal{G})$. In fact, the only differences from the case treated in the proof of Theorem 1 ($length(\pi_0) = length(\mathcal{G})$) is the initialization of π and the definition of the Ω-goal set. This, however, does not affect the correctness of the proof which remains valid also for the cases (a) and (b).

Local Search for Plan Adaptation

In this section we propose a method for plan adaptation that combines ADJUST-PLAN and some local search techniques for planning graphs which we introduced in (Gerevini and Serina 1999). We start outlining the framework for planning using local search, and then we describe the combined method.

Planning through Greedy Action Graphs

Within this framework the problems of generating a new plan and of adapting an existing plan are formulated as search problems where the elements of the search space are particular subgraphs of a planning graph called *Action Subgraphs*.[13] Each of these subgraphs represents a partial plan under generation or adaptation. Typical operators for moving from a search state to the next one are some modifications of the current subgraph corresponding to adding (or deleting) some actions to (from) the partial plan represented by the subgraph.

The main difference between plan adaptation and plan generation concerns the initial action subgraph. In plan adaptation the initial subgraphs is constructed using an existing plan π_0 for an old problem and the planning graph $\mathcal{G}$ of the new problem, while in plan generation the initial subgraph can be any subgraph of the planning graph. In particular, in plan adaptation first we build the planning graph $\mathcal{G}$ of the new problem Π, and we identify the actions of π_0 that are applicable for solving Π (as described in the previous section). Then, starting from the action subgraph $\mathcal{A}_0$ of $\mathcal{G}$ corresponding to the applicable actions of π_0, we run a local search process. This process revises $\mathcal{A}_0$ through the iterative application of the graph modifications until it becomes a *solution subgraph* for Π, i.e., an action graph representing a valid plan for Π.

At any step of the search process the set of actions that can be added or removed is determined by the set of inconsistencies that are present in the plan represented by the current action subgraph. Each inconsistency determines a set of graph modifications that can eliminate it. Two simple types of modifications have been implemented in GPG: the extension of the subgraph to include a new action node of $\mathcal{G}$ (and the relevant edges), and the reduction of the subgraph to remove an action node (and the relevant edges). The study of other types modifications exploiting the semantics of the graph is in progress.

Given a (randomly chosen) inconsistency in the current subgraph $\mathcal{A}$, the "best" modification is chosen according to an *action-cost function* C evaluating the cost of adding or removing an action. Specifically, the cost $C([a],\pi)^i$ of inserting the action node $[a]$ representing the action a, and the cost $C([a],\pi)^r$ of removing $[a]$ are:

$$C([a],\pi)^r = \alpha^r \cdot p(a,\mathcal{A}) + \beta^r \cdot me(a,\mathcal{A}) + \gamma^r \cdot sup(a,\mathcal{A})$$
$$C([a],\pi)^i = \alpha^i \cdot p(a,\mathcal{A}) + \beta^i \cdot me(a,\mathcal{A}) + \gamma^i \cdot unsup(a,\mathcal{A})$$

where $me(a,\mathcal{A})$ is the number of action nodes in $\mathcal{A}$ which are mutually exclusive with $[a]$, $p(a,\mathcal{A})$ is the number of precondition facts of $[a]$ which are not supported, $unsup(a,\mathcal{A})$ is the number of unsupported precondition facts in $\mathcal{A}$ that become supported by adding a to $\mathcal{A}$, and $sup(a,\mathcal{A})$ is the number of supported precondition facts that become unsupported by removing a.[14]

By appropriately setting the parameters (α, β, γ) of the cost function we can implement various heuristics for local search algorithms, some of which are described in (Gerevini and Serina 1999). These methods can be specialized for solving plan adaptation tasks by taking into account the "quality" of the desired adapted plan. For example, in principle it is possible to design heuristics that constrain the adapted plan to contain certain actions of the original plan, or that contain a maximum number of changes with respect to the original plan. However, in this paper we do not consider these issues, since our main interest is on the efficiency of plan adaptation.

Combining Local and Systematic Search

In (Gerevini and Serina 1999a) we show that the local search method can be very efficient when the revised problem admits a solution involving no more time steps than the original plan. However, its performance can significantly depend on the settings of the heuristic parameters, that can be "tuned" by the user. On the other hand, as shown in the next section, ADJUST-PLAN can efficiently solve adaptation tasks requiring a number of time steps higher than the time steps of the original plan, but it can generate adapted plans involving more time steps than necessary (especially when there exists a solution with the same number of time steps as the original plan).

The local search technique and ADJUST-PLAN can be integrated to combine the relative advantages and overcome their limitations. The integrated method, LS-ADJUST-PLAN, that we have implemented and tested in GPG performs local search for (a) computing a compact solution (adapted plan) involving no more time steps that the initial action subgraph, or (b) preprocessing the input plan for reducing the number of its inconsistencies before running ADJUST-PLAN. In particular, LS-ADJUST-PLAN performs the following steps:

1. run the local search technique for at most `t1` CPU-seconds. If a solution is not found, then
2. run ADJUST-PLAN for at most `t2` CPU-seconds. If a solution is not found, then
3. run the local search technique to compute a *quasi-solution*, i.e., an action subgraph representing a plan with at most `k` inconsistencies;
4. run ADJUST-PLAN using as input the quasi-solution computed by previous step.

In step 1 the size of the planning graph is kept constant, and hence any solution that can be found involves no more time steps than the initial action subgraph. On the contrary, during the computation of a quasi-solution (step 3) the number of levels in the actions subgraphs

[13] An action subgraph $\mathcal{A}$ of a planning graph $\mathcal{G}$ is a subgraph of $\mathcal{G}$ such that if a is an action node of $\mathcal{G}$ in $\mathcal{A}$, then the fact nodes of $\mathcal{G}$ corresponding to the preconditions and positive effects of a are also in $\mathcal{A}$, together with the edges of $\mathcal{G}$ connecting them to a (Gerevini and Serina 1999).

[14] A fact node q at level i in $\mathcal{A}$ is *supported* if either at level $i-1$ of $\mathcal{A}$ there is an action-node that is connected to q by an add-effect edge, or $i = 0$ (Gerevini and Serina 1999).

is automatically increased of one level each time a predefined number of search steps is exceeded (initially the new level contains only noop-actions).
In the last step ADJUST-PLAN can be run until it finds a solution, or it detects that the plan cannot be adapted. Alternatively, we can impose a CPU-time limit to ADJUST-PLAN (`max-adjust-time`), after which we repeat steps 3 and 4 to produce another quasi-solution that might be easier to be repaired by ADJUST-PLAN. GPG uses some default values for `t1`, `t2` and `k`, that can be modified by the user.

Experimental Results

In this section we give the results of some experiments aimed at testing the efficiency of ADJUST-PLAN and LS-ADJUST-PLAN. We designed 255 modifications of some known test problems in the domains Rocket, Logistics and Gripper.[15] Figure 4 shows the CPU-time required by our techniques for adapting an existing plan for a known problem to solve several modifications of it. In addition, the figure gives the CPU-time for solving the modifications by replanning from scratch using IPP (version 3.3).

In each replanning problem the Ω-goal set of ADJUST-PLAN was computed according to Definition 5, if the subplan following the replanning window was valid, and according to Definition 4 otherwise. (We also tested the use of an empty Ω-goal set, but as expected the performance of GPG was on average less satisfactory.) The search limits `t1` and `t2` of LS-ADJUST-TIME were set to 2 and 4 seconds respectively, while `max-adjust-time` was set to 32 seconds, and the maximum number `k` of inconsistencies in a quasi-solution to 4. The tests were conducted on PC Pentium II 400 MHz with 256 Mbytes. When IPP was not able to find a solution, the figures plot the value of a CPU-time limit (1000 or 10000 seconds), that was exceeded by IPP, or not reached by IPP, because it ran out of memory. The times for LS-ADJUST-PLAN are average times over 10 runs. The local search was run using the heuristic *T-Walkplan* (Gerevini and Serina 1999) with the following parameter settings: tabu-list length 20; noise 0.2; α^r, β^r and γ^i −0.3; α^i and β^i 1; γ^r 1.5.

Each problem modification is named using a number, and is a variant of a known test problem. For each variant the same plan solving the original problem was given as input to the adaptation process. Each variant contains few changes in the facts of the initial or final state(s) of the original problem, making the input plan not valid for solving the revised problem.[16]

Logistics_a/b/c are three problems introduced by Kautz and Selman (1996) for the Logistics domain (Veloso 1994), which was studied also by Veloso as a plan adaptation test domain. In this domain there are several cities, and at each city there are several locations (e.g., post offices and airports). Some trucks can be used for carrying packages within the same city, and some airplanes can be used for carrying packages between different cities. Typical goals of a planning problem consist of having some packages delivered to some location, and typical problem modifications consist of having a different initial or final location for some package(s). For example, in Logistics_a problem 21 corresponds to a change of the initial position of "package1" from the location "pgh-po" to the location "la_airport", which requires many changes to the original plan; problem 13 corresponds to a change of the goal position of "package1" from "bos-po" to "la_po", which requires at least changing four actions in the plan. A few modifications concerns availability of resources. For example, in problems 41–45 of Logistics_b/c one of the airplanes was removed. These problems require significant changes to the original plan.

Rocket_a and Rocket_b are two problems introduced by Kautz and Selman (1996) in the Rocket domain (Veloso 1994), which has several similarities with Logistics; while Gripper_10/12 are two problems in the Gripper domain, and were taken from IPP's package. In the Gripper domain we have a robot with two arms that has to move several balls located in certain rooms to some other rooms. Here the problem modifications were obtained considering an additional room and changing the initial or final positions of the balls.

In general, the results in Figure 4 show that adapting a plan using our techniques is much faster than a complete replanning (up to four orders of magnitude). For example, the variant 10 of Logistics_a was solved by ADJUST-PLAN and LS-ADJUST-PLAN in 0.28 and 0.29 seconds respectively, while IPP required 4686 seconds.

One major reason is that very often the replanning (sub)problems defined by the replanning windows are much easier than the complete replanning problem considered by IPP, and can be solved by shorter subplans. In general, the performance of ADJUST-PLAN depends on the number of replanning windows that are considered, on their size, and on the hardness of the corresponding replanning problems. Among these factors the first two seems to be less crucial than the third one. In fact, both the problem variants 4 and 17 of Logistics_a were adapted using just one replanning window of length 5, for which plans involving the same number of time steps were computed. However, the variant 17 required much more CPU-time than the variant 4 (almost the same as IPP), because for IPP the replanning problem of the variant 17 was much more difficult than the replanning problem of the variant 4.

In general, ADJUST-PLAN produced adapted plans involving more time steps than LS-ADJUST-PLAN, which often computed optimal plans (see Figure 5). On the other hand, the CPU-time of LS-ADJUST-PLAN can be higher than the CPU-time of ADJUST-PLAN (though it is still significantly lower that the CPU-time of IPP). This is especially true in the results concerning Gripper_10/12, where the local search in LS-ADJUST-PLAN could never find a solution, which was always found by

[15] We used the formalization provided in IPP's package: `http://www.informatik.uni-freiburg/~koehler/ipp.html`.

[16] The formalizations of the problems that we tested are available via anonymous FTP from `ftp.ing.unibs.it`, file `/home/gerevini/adjust-problems.tar`.

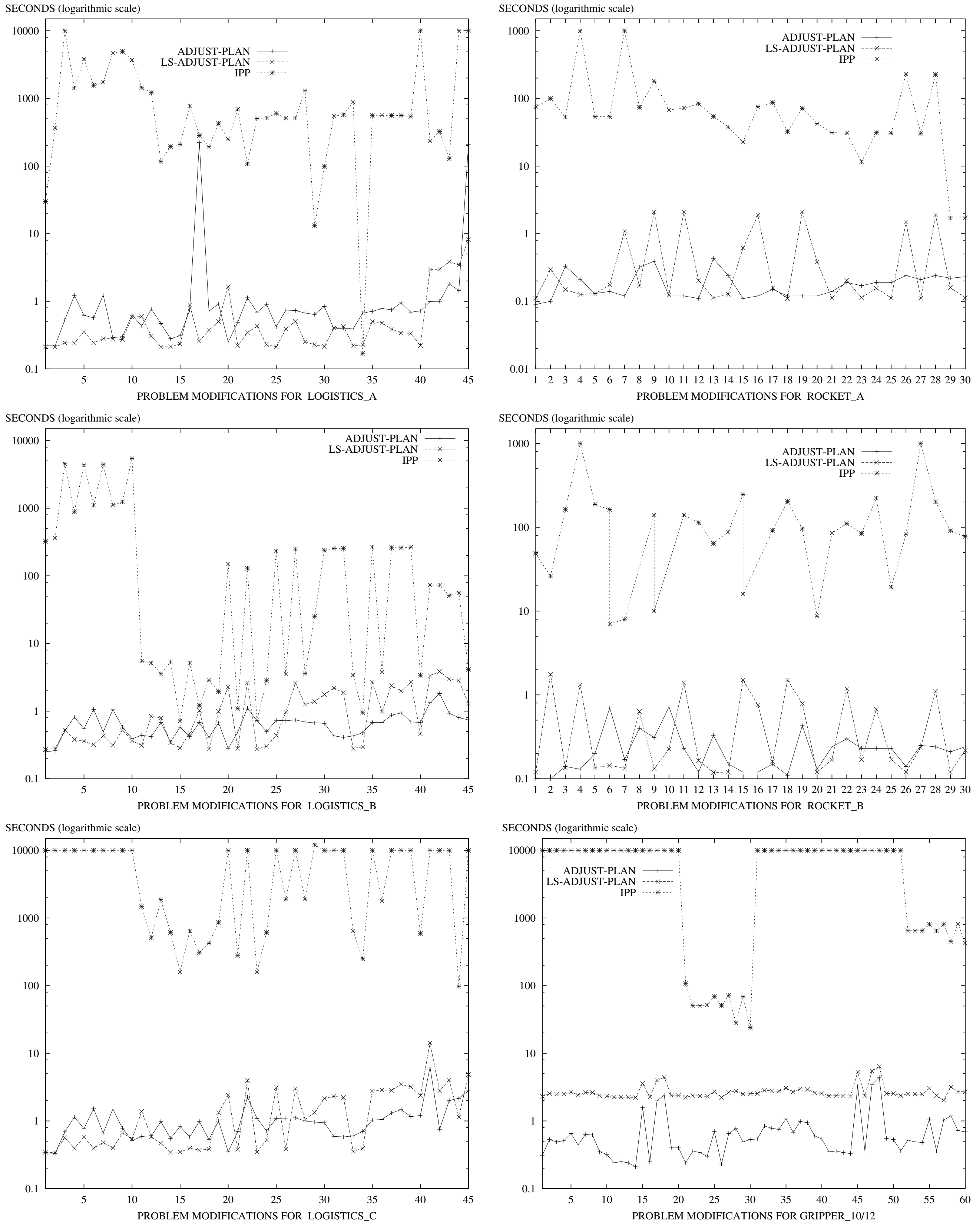

Figure 4: CPU-seconds (logarithmic scale) required by ADJUST-PLAN, LS-ADJUST-PLAN and IPP(3.3) for solving 255 problems derived by modifying the initial or goal state of Logistics_a/b/c, Rocket_a/b and Gripper_10/12.

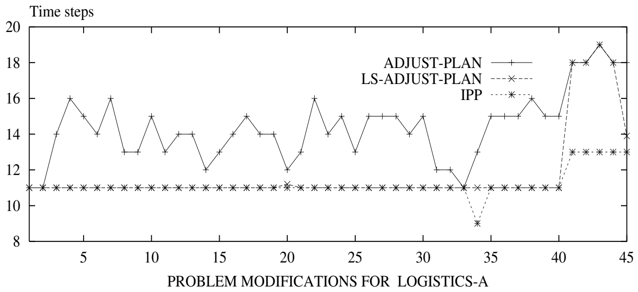

Figure 5: Time steps in the plans computed by ADJUST-PLAN, LS-ADJUST-PLAN and IPP for solving 45 modifications of Logistics_a. The number of time steps for LS-ADJUST-PLAN is almost always the same as for IPP.

the first run of ADJUST-PLAN applied to the original plan (step 2 of LS-ADJUST-PLAN). However, in other cases the local search was very useful, because either it computed a compact solution very efficiently, or it produced a quasi-solution that was much easier to repair for ADJUST-PLAN than the original plan (e.g., modification 17 of Logistics_a).

Finally, we believe that our plan-adaptation can have significant computational advantages with respect to a complete replanning, independently from the particular replanning algorithm. Preliminary results concerning the use of BLACKBOX (Kautz and Selman 1999) for solving the problems of Figure 4 indicate that our plan adaptation approach still performs much more efficiently than a complete replanning.

Conclusions

We have presented some techniques for efficient domain-independent plan adaptation in the context of the planning graph representation, which is adopted by most of the current fastest plan generation systems. Experimental results using GPG show that adapting a plan using our approach can be dramatically more efficient than a complete replanning (up to four orders of magnitude faster).

Further work includes the development of a time-step optimization process for plans computed by ADJUST-PLAN. This process can be seen as an "anytime" method providing a succession of improved plans, each of which involves fewer time steps than the previous plan. The more CPU-time is dedicated to this process, the shorter is the length of the plan computed, up to reach the optimal length (Serina 2000).

Other approaches have been proposed in the literature, including Kambhampati and Hendler's PRIAR system (1992), Veloso's PRODIGY/ANALOGY system (1994), and Hanks and Weld's SPA system (1995). A major difference between GPG and these systems is the underlying planning algorithm and data structures. PRIAR is based on a hierarchical nonlinear planner; PRODIGY/ANALOGY is based on a mean-ends analysis backward-chaining nonlinear planner, performing state-space search; and SPA is based on causal-link partial-order planning. Another difference concerns the input information. While GPG requires only a very simple plan description that could be easily obtained from the output of many planners (or even written by hand), the other mentioned systems use 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.

Our techniques currently require the construction of a planning graph, which for large planning domain can be computationally expensive. We are currently investigating a new version of ADJUST-PLAN using weaker, but computationally more efficient data structures. Other current and future work includes the study of further local search heuristics specialized for plan-adaptation, further techniques for determining replanning goals, additional experimental tests (including a comparison with the performance of other systems).

Acknowledgments

This research was supported in part by "Progetto cofinanziato MURST CERTAMEN". We thank the anonymous referees for their helpful comments.

References

Blum, B., and Furst, M. L. 1995. Fast planning through planning graph analysis. In *Proc. of IJCAI-95* , 1636–1642.

Ferguson, G., and Allen, J. 1996. TRAINS-95: Towards a mixed-initiative planning assistant. In *In Proc. of AIPS-96*, 70–77. AAAI/MIT Press.

Ferguson, G., and Allen, J. F. 1994. Arguing about plans: Plan representation and reasoning for mixed-initiative planning. In *Proc. of AIPS-94*, 13–15. AAAI/MIT Press.

Gerevini, A., and Serina, I. 1999. Fast planning through greedy action graphs. In *Proc. of AAAI-99*, 503–510.

Gerevini, A., and Serina, I. 1999a. Plan generation and adaptation using local search techniques for planning graphs. In *Proc. of CP-AI-OR-99*.

Hanks, S., and Weld, D.S. 1995. A domain-independent algorithm for plan adaptation. *JAIR*, 2, 319–360.

Kambhampati, S., and Hendler, J.A. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55, 193–258.

Kautz. H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. of AAAI-96*, Portland, OR. AAAI/MIT Press.

Kautz, H., and Selman, B. 1999. Unifying sat-based and graph-based planning. In *Proc. of IJCAI-99*.

Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In *Proc. of ECP-97*, 273–285. Springer Verlag.

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

Serina, I. 2000. Generazione ed Adattamento di Piani attraverso Grafi di Pianificazione. *Ph.D Thesis* (in Italian). Universitá di Brescia, Italy. Forthcoming.

Veloso, M. 1994. Planning and learning by analogical reasoning. LNAI 886 Springer-Verlag.