

Modification Strategies for SAT-based Plan Adaptation

Roman van der Krogt
Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
roman@4c.ucc.ie

Abstract

Planning, the generation of a course of action to achieve a set of goals, is an important technique in the development of intelligent agents. Heretofore, planning has been largely considered as a one-shot problem. However, in practice, we are often dealing with situations in which an existing plan has to be adapted. Not only might we be facing a dynamic environment that requires a plan to be repaired, but it may also be that we recognise the new planning problem as being similar to one that we have solved before (i.e. case-based planning).

This paper investigates a plan adaptation framework based on SAT-encodings of the planning problem. Compilation techniques have been very successfully applied to planning, as evidenced by their success in recent planning competitions. So far, however, such techniques have not been used for plan adaptation purposes. This paper explores whether it is feasible to modify the generated SAT instances such as to encode information that was extracted from the solution to the original planning problem.

1 Introduction

Planning, the ability to construct a course of action to achieve desired goals, is an important part of intelligent agents. In planning problems the agent usually has at its disposal (*i*) a description of that part of the world that is relevant, (*ii*) a description of the goals it wants to attain, as well as (*iii*) a description of the actions it can undertake. The task then is to find a (partially ordered) sequence of actions that, when executed, brings the world from its current state into a state in which the agent has attained its goals. However, this approach to planning may not be applicable in some real-world situations.

First of all, this type of planning systems rely on the tacit assumption that planning problems always can be solved off-line: the goal and the initial state are assumed to remain unchanged. For a large number of domains, however, this assumption does not hold due to the dynamical nature of planning problems. For example, in typical transportation problems, planners cannot rely on plans that are constructed off-line: due to e.g., traffic accidents, broken equipment and last-minute changes in orders, the feasibility of the plan can

be seriously affected by a change in the initial state or in the conditions that need to be fulfilled during the execution of the plan. Hence, in these cases, a practical planning approach should also pay attention to replanning to repair a plan that has become infeasible, possibly already during the planning phase. *Plan repair* methods (such as e.g. presented in [11, 34]) are specialised methods to deal with this situation.

Secondly, most planning systems assume that a planning agent has to start from scratch. Often, however, this assumption is not realistic: agents are able to use results of their previous planning experiences, or knowledge given to them by a (human) domain expert and therefore use one or more existing –maybe incomplete or not completely suitable– plans Δ_0 as a starting point. Such initial plans may be transformed and combined into a suitable plan. This approach to planning is called *case-based planning* [29].¹

These are two important reasons why the standard approach to planning, i.e., to start from an empty plan, seems to be just a limiting case of standard practice and usually needs to be generalized to include the adaptation of existing plans. One of the techniques that attempts to overcome the problems mentioned above is that of *plan adaptation*. Plan adaptation tries to adapt an existing plan that was constructed for a different (but similar) problem.

Intuitively, plan adaptation might be perceived as being more efficient than regular planning, since we can reuse part of the existing plan. However, theoretical results show that this is not the case [26]. In fact, under certain conditions it may actually be *harder* to adapt a plan than to construct a new plan from scratch. The most important of these conditions is that a *minimal* plan modification is sought. In practice, plan adaptation systems relax this requirement and do not usually guarantee minimal adaptations. Under these circumstances, plan modification systems are often able to achieve significant speed-up over planning from scratch, while still producing efficient, yet similar, plans (for example, see the results reported in [9, 11, 34]). For this reason, plan modification techniques have remained an active topic of research in the planning community, in the context of both plan repair and case-based planning methods.

This paper investigates a plan adaptation framework based on SAT-encodings of the planning problem. Compilation techniques have been very successfully applied to planning, as evidenced by their success in recent planning competitions [14, 22]. So far, however, such techniques have not been used for plan adaptation purposes. One reason for this is that online methods for SAT, CSP and other approaches that are used to compile the problem into are still an active topic of research in those communities and not generally available. The aim of this paper is to investigate whether it is feasible to compile plan adaptation problems in such a way that standard (i.e. offline) solvers can be used to efficiently solve this type of problem as well. The reason for choosing Satisfiability as our compilation technique, and not, for example, CSP or ILP, is a pragmatic

¹The retrieval of a suitable starting plan is an interesting problem in itself. In this paper however, we focus on the successive step of adapting the plan.

one: easy access to the source code of a successful solver, SatPlan 2006 [19].²

The remainder of the paper is organised as follows. First we present a short overview of related plan adaptation methods and planning methods based on the encoding of planning problems. Then, we present a number of strategies to adapt SAT-encodings to solve the plan adaptation problem in Section 3 and present experimental results of these strategies in Section 4. We finish the paper with conclusions and recommendations for future work.

2 Background

2.1 Planning

Planning is the process of generating a sequence of (possibly partially ordered) actions that achieves a given set of goals [25, 30]. Usually, this process takes as its input a description of the (unique) initial state, a description of the (possibly many) desirable goal states and the operators the agent is allowed to execute.

Definition 1. Following [4] we define a (propositional) planning instance as a tuple $\Pi = \langle \mathcal{P}, \mathcal{O}, I, G \rangle$, where

- \mathcal{P} is a finite set of ground atomic formulae, the *conditions*,
- \mathcal{O} is a finite set of (ground) *operators*, where each operator $o \in \mathcal{O}$ has the form $o : o^+, o^- \Rightarrow o_+, o_-$, where
 1. $o^+ \subseteq \mathcal{P}$ are the *positive preconditions*, i.e. the preconditions that must be true for the operator to be executable;
 2. $o^- \subseteq \mathcal{P}$ are the *negative preconditions*, i.e. the preconditions that must be false for the operator to be executable;
 3. $o_+ \subseteq \mathcal{P}$ is the *add-list*, i.e. the conditions that become true by executing the operator; and
 4. $o_- \subseteq \mathcal{P}$ is the *delete-list*, i.e. the conditions that become false by executing the operator.
- $I \subseteq \mathcal{P}$ is the description of the *initial state*, and
- $G = \langle G_+, G_- \rangle$ is a specification of the *goals*, where $G_+ \subseteq \mathcal{P}$ lists the *positive goals* and $G_- \subseteq \mathcal{P}$ lists the *negative goals*. The positive goal specification lists the conditions that must be achieved. The negative goal specification specifies which conditions must be avoided.

²SatPlan 2006 may be downloaded from <http://www.cs.rochester.edu/u/kautz/satplan/index.htm>

Given a description of a planning problem Π , the task is to find a (possibly partially ordered) sequence of operators from \mathcal{O} , such that when the agent starts executing this sequence in the initial state I , the resulting state after executing all actions satisfies the goal conditions G . More formally, we consider a *plan* Δ to be a sequence of parallel plan steps $\langle \{o_1, \dots, o_k\}^1, \{o_{k+1}, \dots, o_l\}^2, \dots, \{o_m, \dots, o_n\}^T \rangle$ where o_1, \dots, o_k are performed at time step 1, etc. A *linearisation* of such a plan Π is a list of actions $\langle o'_1, \dots, o'_n \rangle$ consisting of a permutation of the actions o_1, \dots, o_k , followed by a permutation of the actions o_{k+1}, \dots, o_l , etc.³

A plan Δ is a solution to a problem Π , if the *result* of the application of *all* linearisations of Δ to I leads to a state S satisfying the goals G .

Definition 2. (Adapted from [4].) Let $\Lambda = \langle o'_1, \dots, o'_n \rangle$ be a linearisation of $\langle \{o_1, \dots, o_k\}^1, \{o_{k+1}, \dots, o_l\}^2, \dots, \{o_m, \dots, o_n\}^T \rangle$. The result of applying Λ to a state S is defined as follows.

$$\begin{aligned} \text{Result}(S, \langle \rangle) &= S \\ \text{Result}(S, \langle o' \rangle) &= \begin{cases} (S \cup o'_+) \setminus o'_- & \text{if } (o')^+ \subseteq S \text{ and } (o')^- \cap S = \emptyset \\ \perp & \text{otherwise} \end{cases} \\ \text{Result}(S, \langle o'_1, o'_2, \dots, o'_n \rangle) &= \text{Result}(\text{Result}(S, \langle o'_1 \rangle), \langle o'_2, \dots, o'_n \rangle) \end{aligned}$$

If for all linearisations Λ of Δ it holds that $G^+ \subseteq \text{Result}(I, \Lambda)$ and $G^- \cap \text{Result}(I, \Lambda) = \emptyset$, we say that Δ *satisfies* Π , denoted by $\Delta \models \Pi$.

We shall refer to the planning task described above as the PLANSAT problem. This problem is defined as follows:

PLANSAT Given an instance of the planning problem $\Pi = \langle \mathcal{P}, \mathcal{O}, I, G \rangle$, does there exist a plan Δ such that $\Delta \models \Pi$?

More informally, we are asking ourselves the question whether there exists a sequence of actions that we can start executing right now (in the current state) and that will bring us to a situation in which we have achieved our goals. In practice, we are of course interested in a slightly different question: does there exist a plan, *and if so, what does such a plan look like?*

Initially, planning research revolved around special purpose algorithms. These were shown to be instances of a common “template” by Kambhampati [15, 16], who called this the *refinement planning* approach. Around 1996, however, Kautz et al. ([18, 20]) started advocating the use of SATISFIABILITY [10, p. 39] solvers in planning, resulting in the BlackBox planner [17]. Their idea was that since SAT solvers had become so

³This is the common semantics of parallel plans. It requires that parallel actions do not interfere.

efficient, even on large instances, a system making use of such solvers can translate and solve planning problems faster than special purpose planning algorithms can find a solution.

A straight-forward way to encode a planning problem as a Satisfiability problem, is to impose a bound k on the makespan of the plan. We now introduce propositional variables P_i^t , denoting that proposition P_i is true at timestep t , and action variables A_i^t to denote that action A_i is executed at time t . Each of those variables is boolean, and we create variables for all propositions P_i and all actions A_i , over all timesteps $0 \leq t \leq k$. It is clear that a plan now corresponds to a particular instantiation of the variables, reflecting the actions that are executed and the states that are visited. Not all instantiations correspond to valid plans, however. Therefore, clauses are introduced to describe valid relationships between the variables. For example, the original BlackBox encodings feature clauses to state that an action may only be executed if its preconditions hold (i.e. of the form $A_i^t \rightarrow P_j^t$, where A_i^t represents an action and P_j^t corresponds to a precondition of that action), and that the effects of an action take place upon execution (i.e. clauses of the form $A_i^t \rightarrow P_j^{t+1}$, where P_j^{t+1} is an effect of the action represented by A_i^t). For a full overview of BlackBox' clauses see [20].

It is the nature of variables and the clauses over these variables that determines the differences between the different SAT-based planners. The SatPlan 2006 planner we have used for our experiments, employs various encodings based on the Graphplan [3] planning graph [19]. Some of these directly encode the add and delete relations that are present between actions in subsequent layers, removing the need for the propositional variables P_i^t . This leads to smaller SAT instances, which may lead to a better performance. In our section on experimental results, we show the influence of the different encodings that SatPlan supports on the performance.

The encodings from the SatPlan family are not the only ones. Ernst et al. [8] extended the work on Blackbox by proposing and analysing a framework that results in a number of different encodings (including the BlackBox encodings), depending on the parameters. More recently, Rintanen et al. [27] show how to obtain more efficient encodings by either further restricting, or relaxing the common semantics for parallel plans. They also investigate how multiple SAT solvers can work in parallel to concurrently test plans of different lengths. MaxPlan (which uses max-SAT to model their plans) introduces more powerful mutex constraints that reason over actions occurring at different timesteps [5], and also uses learning techniques to carry over information between subsequent runs (of increasing plan length) of the SAT solver [36].

Satisfiability is certainly not the only method that has been employed. Other formalisms that have been used include integer programming (e.g. [31, 32]) and constraint satisfaction (e.g. [2, 23, 35]). In this paper we focus on Satisfiability, however, as it has received the most attention and is therefore the most mature.

2.2 Plan Adaptation

Unfortunately, planning as we have introduced above, does often not apply to real world situations. The reason for this is that planning is regarded as a one-shot activity: a planning system is presented with a planning problem, solves it, and that's the end of the story. This, of course, is usually not the case. There are two main objections to this view.

Firstly, we can often not assume that the planning problem is static. Unless the plan is executed in a very controlled environment, it may well be that the assumptions for which the plan was constructed (i.e. the initial state) may change due to circumstances beyond our control (e.g. the behaviour of other agents). Thus, an agent might be faced with occasions in which it needs to alter parts of its plan to be able to still attain its goals. One way would be to simply perform a complete replanning from the current state. However, *plan repair* may often be more efficient, since a large part of the plan usually is still valid. On top of this, in many problem domains it may be quite costly to change your whole plan, for example because of bookings or commitments to other agents that have been made based on the original plan. Furthermore, in mixed initiative settings (such as presented in e.g. [1]), the user might more easily accept a plan that was repaired (and thus resembles the original plan) over a plan that was created from scratch (and might look entirely different).

Secondly, in production environments it is likely that we are faced with planning problems from the same domain over and over again. In fact, we may be faced not just with problems from the same domain, but the problems themselves might be very similar again and again. Given that we know that planning is a very hard problem to solve (*cf.* [4]), would it not be more sensible to record past plans, and try to reuse plans when we encounter situations that (are very similar to those that) have been encountered before? Indeed, case-based reasoning systems were developed for precisely this reason [21]. Case-based planning [29] is a form of planning that applies case-based reasoning techniques. Case-based planning systems retrieve a plan from memory (which is an interesting problem in itself, but beyond the scope of this research), and then adapt it to suit the new situation.

Notice the similarity in the two cases: we have at our disposal an original plan Δ_0 to a problem Π_0 , and, for reasons of computational efficiency, costs or similarity, we are interested in modifying this plan, such that it is applicable to a new or changed situation. This leads to the following formal problem definition:

PLANMOD Given an instance of the planning problem $\Pi = \langle \mathcal{P}, \mathcal{O}, I, G \rangle$ and a plan Δ_0 that satisfies an instance $\Pi_0 = \langle \mathcal{P}_0, \mathcal{O}_0, I_0, G_0 \rangle$, produce a plan Δ that satisfies Π by *modifying* Δ_0 .

As for planning, it has been shown that special purpose algorithms for solving this problem are largely

instances of the same template, in this case that of *unrefinement planning* [34]. Unlike planning, however, encoding techniques have not been investigated yet. One of the main reasons for this is that online (i.e. dynamic) solvers for SAT, CSP, etc. are not common. Additionally, the online solvers that exist are often for specific cases, rather than general solvers. In the next section, we introduce straight-forward modification strategies to adapt SAT instances to reflect information on how the original plan was built. Such a modification strategy encodes this information in the instance, making it possible to use a standard solver.

3 Modification Strategies

This section presents a number of strategies to adapt a SAT-encoding to reflect the original planning problem. The process we envision is the following. Suppose we have an original planning problem Π_0 , a plan Δ_0 that is a solution to Π_0 and a current problem Π . A SAT-based planner generates a SAT-encoding \mathcal{S}_Π for Π from scratch. This is then combined with information from Π_0 and Δ_0 to produce a new SAT instance \mathcal{S}'_Π , that reflects information from Δ_0 on the common aspects of Π and Π_0 . Notice how our scheme does not rely on a particular solver. Rather we are interested in encoding-neutral modification strategies that work irrespective of (the encoding of) the particular planner, so we can exploit future improved encodings. The strategies below work by recognising actions in the original plan that are present in the encoding. We then add clauses to the encoding to assert those

actions. The difference of the strategies lies in the subset of actions they recognise from the plan. By recognising and asserting more actions, we may reuse more of the original plan, but are also more restricted in the plans we may find. This trade-off is explored in the next section, on our experimental results.

In the discussion below, we consider grounded actions, i.e. actions with their parameters instantiated. The original problem is denoted by $\Pi_0 = \langle \mathcal{P}_0, \mathcal{O}_0, I_0, G_0 \rangle$, the current problem by $\Pi = \langle \mathcal{P}, \mathcal{O}, I, G \rangle$ and the plan that satisfies Π_0 equals $\Delta_0 = \langle \{o_1, \dots, o_k\}^1, \{o_{k+1}, \dots, o_l\}^2, \dots, \{o_m, \dots, o_n\}^T \rangle$. Let o^t denote that an action o is executed at timestep t and let the SAT formula stating that o is to be executed at time t be denoted by $\varphi(o^t)$. In many encodings the latter corresponds to a single variable, but this is not required. Moreover, it may be the case that $\varphi(o^t)$ cannot be expressed in terms of the encoding \mathcal{S}_Π . This is a common situation in graphplan-based encodings, when the planning graph has ruled out o from occurring at time step t . If

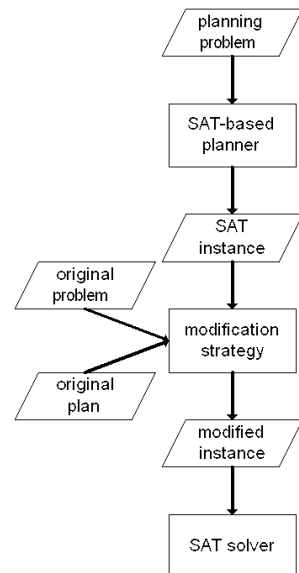


Figure 1: Modification Process

this is the case, $\varphi(o^t) = \top$, which is always true. We let $\varphi(o)$ denote the formula $\varphi(o^1) \vee \varphi(o^2) \vee \dots \vee \varphi(o^T)$, where T is the current planning horizon.

Finally, we introduce some notation for the causal relationships in the plan. We say $o_i^{t_i} \xrightarrow{p} o_j^{t_j}$ if o_i satisfies some precondition p of o_j and there is no action $o_k^{t_k} \in \Delta_0$, $t_i < t_k < t_j$ that also has p as an effect. If it does not matter which particular precondition is involved, we simply write $o_i^{t_i} \rightarrow o_j^{t_j}$. Similarly, we introduce this notation for actions that satisfy the final goal conditions and actions that have preconditions satisfied by the initial state: $o_i^{t_i} \rightarrow g$, $g \in G_0$, holds when $o_i^{t_i} @ t_i$ is the last action having g as effect; and $\iota \rightarrow o_i^{t_i}$ holds when there is no action $o_k^{t_k}$, $k < i$ that has ι as an effect.

The transitive closure of \rightarrow is denoted with \rightsquigarrow , i.e. $o_i^{t_i} \rightsquigarrow o_j^{t_j}$ holds if either (i) $o_i^{t_i} \rightarrow o_j^{t_j}$, or (ii) there exists an action $o_k^{t_k} \in \Delta_0$, $t_i < t_k < t_j$, such that $o_i^{t_i} \rightsquigarrow o_k^{t_k} \wedge o_k^{t_k} \rightarrow o_j^{t_j}$.

Nothing

The most simple strategy is to make no changes at all, i.e. $\mathcal{S}_{\Pi}^{nothing} = \mathcal{S}_{\Pi}$. This corresponds to planning from scratch. It is included as a base case.

Full Assertion

The *Full* strategy asserts all actions $o_i^{t_i} \in \Delta_0$ by including clauses asserting the detected actions to occur at the exact times that they originally occurred. That is,

$$\mathcal{S}_{\Pi}^{full} = \mathcal{S}_{\Pi} \wedge \bigwedge_{o_i^{t_i} \in \Delta_0} \varphi(o_i^{t_i})$$

Notice how this may lead to inefficiencies in the plan (when actions are enforced that are no longer applicable) or even result in unsatisfiable instances (cf. the example below).

A relaxed version of this strategy, denoted by *Full**, does not assert the action at the time it was present in the original plan, but rather asserts that the action takes place within the planning horizon:

$$\mathcal{S}_{\Pi}^{full*} = \mathcal{S}_{\Pi} \wedge \bigwedge_{o_i^{t_i} \in \Delta_0} \varphi(o_i)$$

Helpful Action Assertion

This strategy compares the goals of the original and the new problem. For matching goals, it identifies those actions that contributed to achieving these in the original plan. These *helpful actions* are then asserted at the times that they originally occurred. To be precise, let $G' = G \cap G_0$, i.e. all goals that are common to

Π and Π_0 , and let $H = \bigcup_{g \in G'} \{o_i^{t_i} \in \Delta_0 \mid o_i^{t_i} \rightsquigarrow g\}$, i.e. all actions that contributed to achieving any of the common goals. Then,

$$\mathcal{S}_{\Pi}^{\text{helpful}} = \mathcal{S}_{\Pi} \wedge \bigwedge_{o_i^{t_i} \in H} \varphi(o_i^{t_i})$$

Again, a relaxed version of this strategy allows the actions to occur at any time in the plan. (We will no longer explicitly mention the relaxed version for the remainder of the strategies.) These strategies exploit situations in which goals have been removed from a planning problem. The actions achieving those goals are ignored, whereas the actions achieving the goals that are still valid are asserted.

Supported Action Assertion

This strategy identifies which actions of the original plan can be successfully executed from the new initial state. Formally, let $I' = I_0 \setminus I$, i.e. all propositions that are no longer true, and let $S = \{o_i^{t_i} \in \Delta_0 \mid \nexists \iota \in I' \cdot \iota \rightsquigarrow o_i^{t_i}\}$, i.e. those actions that do not (directly or indirectly) depend on these propositions I' . Then,

$$\mathcal{S}_{\Pi}^{\text{supported}} = \mathcal{S}_{\Pi} \wedge \bigwedge_{o_i^{t_i} \in S} \varphi(o_i^{t_i})$$

Here too, a relaxed strategy relaxes the times at which the actions must occur. This strategy is focused on changes in the initial states. Any actions that can still be executed, are; the others are discarded.

Helpful Supported Action Assertion

This is a combination of the Helpful Action and Supported Action strategies, which asserts actions that are both supported and helpful. This strategy takes into account both changes in the initial state, as well as changes to the goals. Formally, let H be the set as defined under the helpful actions, and let S be the set as defined for the supported actions. Now, let $HS = H \cap S$. Then,

$$\mathcal{S}_{\Pi}^{HS} = \mathcal{S}_{\Pi} \wedge \bigwedge_{o_i^{t_i} \in HS} \varphi(o_i^{t_i})$$

Final Helpful Action Assertion

This strategy is variation on the Helpful strategy. Whereas the Helpful strategies assert all helpful actions, this strategy only asserts the actions directly satisfying goal propositions. To be precise, let $G' = G \cap G_0$, i.e. all goals that are common to Π and Π_0 , and let $F = \bigcup_{g \in G'} \{o_i^{t_i} \in \Delta_0 \mid o_i^{t_i} \rightarrow g\}$, i.e. F consists of those actions that produce the goals. Then,

$$\mathcal{S}_{\Pi}^{final} = \mathcal{S}_{\Pi} \wedge \bigwedge_{o_i^{t_i} \in F} \varphi(o_i^{t_i})$$

Example. To illustrate how these strategies work, consider the following simple transportation planning problem. There are three actions: **move**(l_1, l_2) that moves the one available truck from l_1 to l_2 , **load**(p, l) that loads a package p at location l into the truck and, finally, **unload**(p, l) that unloads from the truck the package p at location l . There are three locations: the airport A , the harbour H and the post office P .

Suppose the original plan involved transporting a box B from H to A , using a plan $\Delta_0 = \langle \mathbf{load}(B, H)^1, \mathbf{move}(H, A)^2, \mathbf{unload}(B, A)^3 \rangle$. The current problem that is to be solved involves the transportation of B from its current location P to its destination A . Without loss of generality, assume we have a graphplan based encoding, where a single variable v_{o^t} is introduced for each grounded action o at each of its possible timesteps t .

If we were to apply the *full* strategy, we would add $\varphi(\mathbf{load}(B, H)^1)$, $\varphi(\mathbf{move}(H, A)^2)$ and $\varphi(\mathbf{unload}(B, A)^3)$ as clauses to \mathcal{S}_{Π} . In the graphplan based encoding, $\varphi(o_i^{t_i})$ evaluates to the singleton clause $v_{o_i^{t_i}}$ for the **move** and **unload** actions. Since there is no variable denoting $\mathbf{load}(B, H)^1$ (as this action is not possible at this timestep), $\varphi(\mathbf{load}(B, H)^1)$ evaluates to \top and can be disregarded. Thus, the Satisfiability problem that is generated is $\mathcal{S}_{\Pi}^{full} = \mathcal{S}_{\Pi} \wedge v_{\mathbf{move}(H, A)^2} \wedge v_{\mathbf{unload}(B, A)^3}$. Notice that this problem does not have a solution (of any length), as no valid plan for the new problem can contain the two enforced actions.

The relaxed strategy *full** would add the following two clauses $v_{\mathbf{move}(H, A)^1} \vee v_{\mathbf{move}(H, A)^2} \vee v_{\mathbf{move}(H, A)^3}$ and $v_{\mathbf{unload}(B, A)^1} \vee v_{\mathbf{unload}(B, A)^2} \vee v_{\mathbf{unload}(B, A)^3}$, to allow the actions to occur at any time. This, too, has no solutions for $T = 3$. However, unlike the situation for the *full* strategy, here there are larger planning horizons that would lead to a (suboptimal) solution.

Finally, the *final* strategy would add $\varphi(\mathbf{unload}(B, A)^2)$, as this is the action achieving the goal (which has not changed).

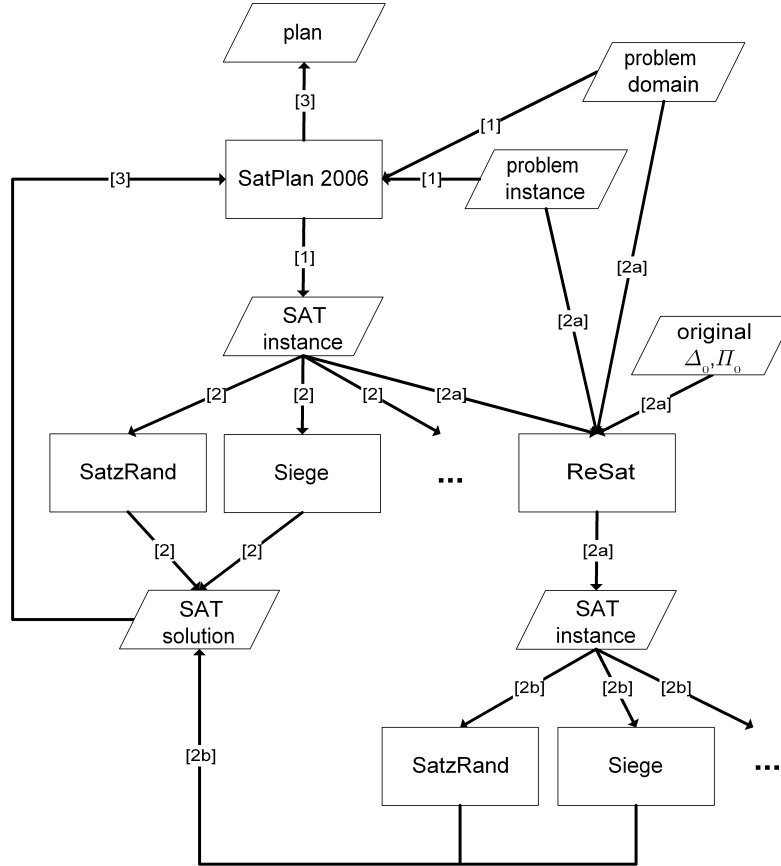


Figure 2: Architecture of SatReplan

4 Experimental Results

4.1 Setup

The strategies above were implemented as an external solver for SatPlan 2006 [19]. This planner supports a range of external SAT solvers by default (BerkMin [12], Jerusat [24], MiniSat [7], Siege [28] and Satzrand [13]) and allows easy integration of new ones. We implemented an additional layer to the system that acts like a solver to SatPlan (albeit with a number of additional parameters). The resulting architecture is shown in Figure 2. The normal operation of SatPlan is a three step process: first, it reads the problem domain and the problem instance and encodes the planning problem as a Satisfiability instance. This is then passed to the SAT solver of choice, which produces a solution to the SAT instance (if the planning problem is feasible, of course). Finally, the SAT solution is translated back into a plan.

In order to adapt a plan, the process starts identically. The planning problem is translated into a SAT instance. This is passed by SatPlan to our ReSat system, as though it were a regular solver. However, ReSat modifies the SAT instance according to the preferred modification strategy and the specification of

the original planning problem and its plan, producing a new SAT instance (step 2a in the figure). This is then solved as normal (step 2b) and the solution is read by SatPlan to reconstruct the solution of the planning problem. Notice that the modular nature of the system leads to a loss of efficiency as problem files have to be parsed several times by different modules. However, we think the benefits in ease of use and adaptability outweigh such concerns, in particular for large problems where most time is spent on the actual solving of the problem.

The experiments we report on were performed using Siege, SatPlan’s default solver. A small sample of problems showed some variation between the different solvers supported, but no major differences when it comes to adapting problems. (Siege performed on average.) Our tests comprise a total of 4 domains. Three domains come from the benchmark problem set of GPG [11]: Logistics, Gripper and Rocket. These problems can be divided into 7 sets (2 each for the gripper and rocket domains, and 3 for logistics). Each set contains variants on the same test problem (thus, a total of 7 base problems are used). Each of the variants has a few changes to the initial state, the goals, or both. (See [33, Appendix D] for a detailed description of the problem instances.) This benchmark set is already a few years old, but is compatible with the version of PDDL that SatPlan supports.

The fourth domain is the Satellite domain from the 2002 and 2004 International Planning Competitions. For each of the 20 problems in the benchmark set, we generated 12 variations. Three of those had some of the initial state changed (with either 10, 25 or 50 percent of the literals changed); three had additional initial state variables, representing additional satellites and equipment (the number of literals was increased by either 10, 25 or 50 percent); six more had either additional goals or changes to goals, in a way similar to the way the initial state was changed. This setup was chosen in order to study the behaviour of the strategies for different types of problems. The fact that the 20 problems grow in size, allows us to look at the asymptotic behaviour of the system. See Appendix A for a discussion on how the problems were generated.

The following two hypotheses will be tested:

Hypothesis 1. *Modifying the SAT instances results in an improvement (in time) over solving from scratch.*

Hypothesis 2. *Modifying the SAT instances results in larger makespans for the non-relaxed strategies. The relaxed strategies do not suffer from this.*

Notice that a comparison with other plan repair systems is not made, as the available systems all optimise for number of actions, rather than make-span. This would make a comparison unfair.

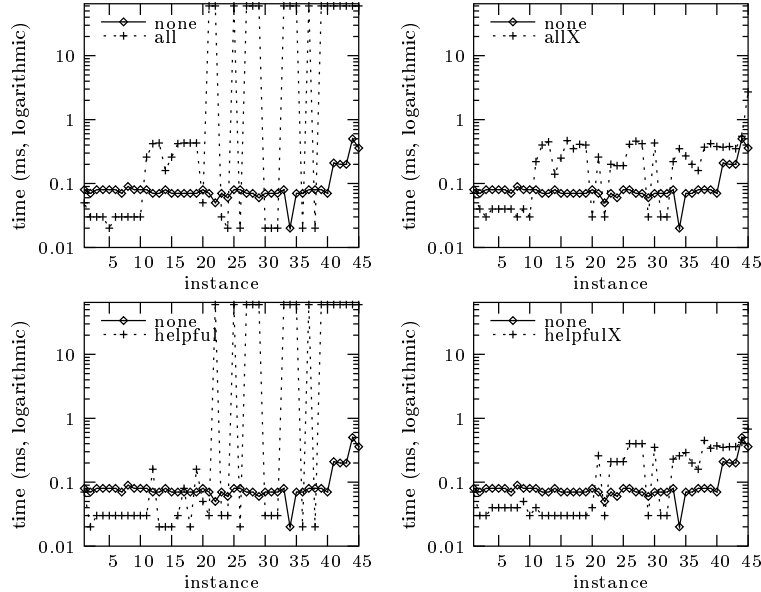


Figure 3: Performance on the *thin gp-based* encoding of the *all* and *helpful* strategies, *logistics-A* set.

4.2 Efficiency

Informally, our first hypothesis states that it takes less effort to solve the modified SAT instances, than it takes to solve the unmodified ones. To investigate this hypothesis, we compare the runtimes of the original SatPlan 2006 (for all four supported encodings) with the runtimes of our SatReplan system, for the various strategies. Each problem was solved 3 times; the average values were then used to make the comparison.

We first consider the GPG benchmark set. Before going into the details, let us first consider a few graphs to get a feel for the data. Figures 3 through 5 show the performance of some of the strategies for a number of encodings and problem domains. Each graph shows the time Siege needs to solve the unmodified instance and the time to solve a modified instance using a specific strategy. The cut-off time was 60 seconds, except for the Gripper problems, where the cut-off was 5 minutes (i.e. 300 seconds). As we can see, the *all*, *helpful*, *supported* and *HS* strategies all produce instances that cannot be solved within the 60 seconds. In fact, it turns out that on those instances the assertions that are made are conflicting, preventing a solution to be found. The relaxed version of these strategies, as well as the *final* and *final** strategies do not suffer from such occurrences. We can also clearly see the benefit of plan adaptation in the results for the Gripper domain. Whereas SatPlan itself does not find a solution for most of the problems, the plan adaptation system using the *final(*)* strategies finds solutions for a much larger number of solutions.

To see whether the the observed differences between the systems are statistically significant, we have used the Chi-Squared test (see, e.g. [6]), following the analysis performed since the 2003 International Planning Competition [22]. In our analysis, we count the number of times each system is the best performer (e.g. the

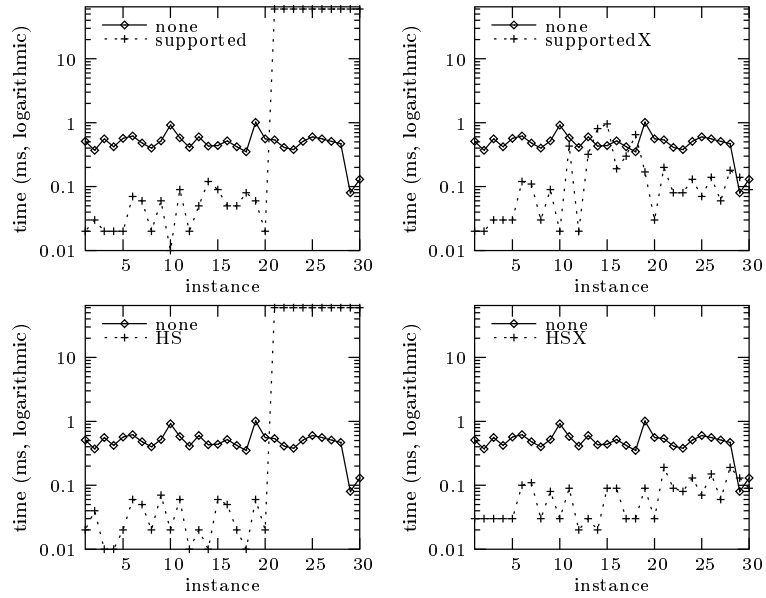


Figure 4: Performance on the *action-based* encoding of the *supported* and *helpful-supported* strategies, *Rocket-A* set.

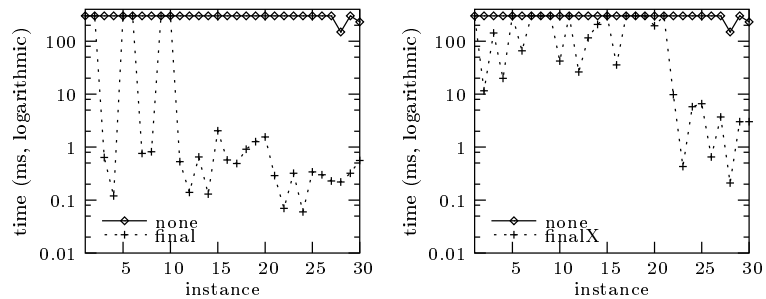


Figure 5: Performance on the *gp-based* encoding of the *final* strategy, *Gripper-10* set.

Table 1: Significantly faster strategies. *Thin gp-based* encoding, *logistics* problems

action-based	none	all	all*	helpful	helpful*	supported	supported*	HS	HS*	final	final*
none	-	-	←	-	↑	-	-	↑	↑	↑	↑
all	-	-	-	↑	↑	-	-	↑	↑	↑	↑
all*	↑	-	-	-	↑	-	↑	↑	↑	↑	↑
helpful	-	←	-	-	-	←	-	-	-	-	-
helpful*	←	←	←	-	-	←	-	-	↑	↑	-
supported	-	-	-	↑	↑	-	-	↑	↑	↑	↑
supported*	-	-	←	-	-	-	-	↑	↑	↑	↑
HS	←	←	←	-	-	←	←	-	↑	↑	-
HS*	←	←	←	-	←	←	←	←	-	-	←
final	←	←	←	-	←	←	←	←	-	-	←
final*	←	←	←	-	-	←	←	-	↑	↑	-

fastest or with the shortest makespan). If the systems are not significantly different, we can expect that the number of times each system outperforms the others is about equal. If the frequencies are far from equal, however, we can reject the hypothesis that the systems perform equally well. This is exactly what the Chi-Squared test will tell us. When comparing runtimes, we consider one method faster than another, if it is at least 10% faster.⁴ We only take into account the time the solver takes. This is to nullify the overhead caused by the modular architecture, which could be removed in a production system. For makespan, we simply consider the makespan to determine the winner.

Before drawing more general conclusions, let us consider some specific data. Table 1 shows the results of pair-wise χ^2 tests for all modification strategies, for the *thin-gp based* encoding on the *logistics* set. (The *none* strategy is the standard SatPlan 2006 result.) Each cell in the table relates the strategies in its row and column. If we can not declare a statistically significant difference between the strategies (i.e. $p \geq 0.05$), the cell contains a '-'. Otherwise, the arrow in the cell points towards the better (i.e. faster) strategy. For example, the column marked 'final' has most cells marked \uparrow . This means that it is (pairwise) better than most of the other strategies. Results for the other GPG problem sets and encodings are similar.

We summarise the results by looking at all strategies at once. Table 2 shows the results for the different domains. Each cell is the number of times that the particular strategy was significantly better than all others. From this table, we can see that for all four encodings that SatPlan 2006 is able to generate, the strategies are significantly different from each other. Also, we can observe that the *final* and *final** strategies often produce the best results, although the *all* and *helpful* strategies show good results on the *action-based* encoding. The *none* strategy (i.e. planning from scratch) is among the worst performers.

⁴That is, method m_1 taking t_1 seconds is considered faster than m_2 taking t_2 seconds, iff $1.10 \times t_1 < t_2$.

Table 2: χ^2 test for the whole population of strategies for all domains (overall wins)

encoding		none	all	all*	helpful	helpful*	supported	supported*	HS	HS*	final	final*	χ^2	p
logistics	action-based	0	12	0	19	3	1	4	3	0	31	11	128.4048	< 0.01
	gp-style action-based	1	2	0	2	1	3	0	5	0	31	6	173.5294	< 0.01
	gp-based	1	0	0	7	0	5	0	10	1	21	5	91.24	< 0.01
	thin gp-based	1	2	0	8	0	2	0	8	1	21	5	90.4167	< 0.01
rocket	action-based	0	1	0	5	0	1	2	10	0	7	4	41.8667	< 0.01
	gp-style action-based	0	0	0	0	0	0	1	0	3	7	5	41.75	< 0.01
	gp-based	0	0	1	1	2	2	1	3	1	5	5	16.1905	0.09
	thin gp-based	0	0	0	3	0	2	2	4	3	8	6	27.7857	< 0.01
gripper	action-based	0	6	0	17	2	1	4	1	13	9	3	63.0357	< 0.01
	gp-style action-based	1	2	0	7	1	3	0	4	10	12	1	46.1951	< 0.01
	gp-based	1	0	0	9	0	3	0	2	5	11	1	51.1875	< 0.01
	thin gp-based	1	1	0	12	0	3	0	4	6	9	1	48.9189	< 0.01
satellite	action-based	0	47	0	25	0	1	0	1	0	23	0	284.5979	< 0.01
	gp-style action-based	3	1	0	2	1	5	0	4	3	13	9	43.5122	< 0.01
	gp-based	5	1	0	0	1	1	1	3	1	15	15	82.093	< 0.01
	thin gp-based	11	6	0	1	0	5	0	7	2	33	10	134	< 0.01

For completeness, the table also includes the results on the Satellite domain. We will take a more detailed look at those results in the next section, but note that it shows roughly the same pattern when we consider all results from this domain together.

The results from this table shows which of the strategies was able to claim an overall win. That is, how often the strategy was faster than all other strategies (including the *none*, i.e. planning from scratch). The hypothesis we are interested in, however, asks whether each of the strategies is faster than planning from scratch when compared head to head. Therefore, we collected results on how often each of the strategies was faster than planning from scratch. This information can be found in Table 3. It shows, for example, that the *all* strategy was faster in 75 (out of a total of 135) instances on the *logistics* domain, *action-based* encoding.⁵ Again, we can see that the *final* strategy performs particularly well, although the results are less pronounced in the *rocket* domain (where the χ^2 test shows that the differences in the results are not statistically significant). The significance of those numbers is presented in the table as well: bold numbers mean that the strategy performed significantly better; italic numbers mean that the strategy performed significantly worse. We can see that all strategies perform significantly better in the majority of the cases. For the *HS*, *final* and *final** strategies this is even true for all cases. From this, we conclude that the first hypothesis holds, although we do note that there are differences between the strategies (as is to be expected).

⁵Recall that the *logistics* domain features 135 problems, the *rocket* and *gripper* domain each have 60 problems, and the *satellite* domain consists of 240 instances.

Table 3: χ^2 test for the whole population of strategies for all domains (wins over planning from scratch)

encoding		all	all*	helpful	helpful*	supported	supported*	HS	HS*	final	final*	χ^2	p
logistics	action-based	75	39	75	60	76	40	77	67	118	124	94.0732	< 0.01
	gp-style action-based	56	51	78	81	57	67	79	102	125	122	77.7702	< 0.01
	gp-based	55	52	78	84	56	66	79	102	124	131	86.4583	< 0.01
	thin gp-based	56	54	78	86	57	68	79	104	125	130	82.7969	< 0.01
rocket	action-based	41	45	41	51	42	53	42	59	46	60	10.0417	0.35
	gp-style action-based	39	45	46	50	39	52	46	57	50	60	8.8099	0.46
	gp-based	34	35	43	45	35	46	44	55	48	60	14.7978	0.10
	thin gp-based	41	38	45	46	42	48	46	57	50	58	8.0446	0.53
gripper	action-based	20	0	22	1	20	2	21	4	41	12	106.7203	< 0.01
	gp-style action-based	20	0	22	2	20	15	22	17	46	27	77.6387	< 0.01
	gp-based	20	4	22	5	20	18	22	19	45	29	58.7451	< 0.01
	thin gp-based	20	4	22	5	20	17	22	20	46	32	63.0577	< 0.01
satellite	action-based	165	52	156	52	82	42	85	42	150	52	256.9658	< 0.01
	gp-style action-based	87	77	85	79	94	75	95	80	90	94	5.986	0.74
	gp-based	77	80	76	84	91	86	96	88	95	105	8.6515	0.47
	thin gp-based	100	97	104	93	116	73	115	75	124	125	29.9569	< 0.01

Asymptotic Behaviour on the Satellite Domain

The GPG benchmark set features problems of varying nature and it is hard to make out whether one type of problem is more suited to a particular strategy than others. Also, some may scale better than others. We illustrate this behaviour of the system using the final benchmark set: the Satellite domain. We do so along different axes: the size of the problem, as well as the level and type of change. Notice that the size of these problems is much larger than those contained in the GPG set. The cut-off here was set to 30 minutes of CPU time (on an Intel^(R) XeonTM 2.8 Ghz processor), and each solver was allowed 1 Gb of memory. Often, the memory limit was reached while trying to prove the in-existence of a solution. For the purposes of the graphs below, we assigned a runtime of 1800 seconds to those cases. Again, the results are averaged over 3 runs.

As discussed before, we look at four different types of change: changes to the initial state, additions to the initial state, changes to the goals and additional goals. For each type of change, we look at three levels (10, 25 and 50 percent – one can think of those as light, medium and heavy change). We first give an overview of which strategies perform best given the different types of problems. After that, we present graphs relating the best performing strategies with the default SatPlan performance. In order to limit the number of dimensions to the problems, we report here only on the *thin gp-based* encoding.

Table 4 shows the results over the different types of changes. First, we can see that the *final* strategy performs reasonably well again. However, in a number of cases the results are inconclusive. This is especially

Table 4: χ^2 test for the whole population of strategies for all types of changes (overall wins)

	percentage	none	all	all*	helpful	helpful*	supported	supported*	HS	HS*	final	final*	χ^2	p
I change	10%	0	1	0	0	0	0	0	1	0	6	0	44.25	< 0.01
	25%	0	1	0	0	0	0	0	1	1	6	1	34	< 0.01
	50%	0	1	0	0	0	1	0	1	0	9	0	65	< 0.01
I add	10%	0	2	0	0	0	1	0	0	0	0	0	15.3333	0.12
	25%	3	0	0	0	0	0	0	0	0	0	0	30	< 0.01
	50%	6	0	0	0	0	0	0	0	0	0	0	60	< 0.01
G change	10%	0	1	0	0	0	1	0	0	0	0	2	12.5	0.25
	25%	1	0	0	0	0	1	0	2	1	3	1	11.7778	0.30
	50%	1	0	0	0	0	1	0	1	0	4	1	19.5	0.03
G add	10%	0	0	0	1	0	0	0	0	0	0	4	32.4	< 0.01
	25%	0	0	0	0	0	0	0	1	0	2	1	12.5	0.25
	50%	0	0	0	0	0	0	0	0	0	3	0	30	< 0.01

true when we look at the results pertaining to the goals, where we observe that only half of the results are statistically significant. Closer examination of the results showed that for most instances, a number of the strategies showed very similar performance, giving none of the strategies a clear win over all others. This also explains the odd results that were obtained for the enlarged initial states. Here, we see that using planning from scratch is significantly better in 3 and 6 of the instances. For all 9 instances, it was the case that modified SAT instances could not be solved with the available memory, whereas the unmodified instance could be. For all other instances, none of the strategies achieved a clear majority over the other strategies.

Just as we did in the previous section, we also looked at the number of times that each strategy was faster (or slower) than planning from scratch. These results are presented in Table 5. As one can see, none of the strategies performed significantly worse over a set of problems. However, we do see that the number of significant results get less when the problems get more distorted (i.e. when the percentages go up). This is to be expected, as the benefit of plan adaptation becomes less and less, as we can reuse less and less of the original plan.

Finally, we end this section looking at the behaviour of the strategies as the problems get harder. Figure 6 shows graphs comparing the runtimes of the *final* strategy with those of planning from scratch. We can see that in general, the *final* strategy is able to solve harder problems, which means that it scales slightly better. However, on the larger problems, memory becomes the dominant factor and the difference is lost. Notice the six instances of which we spoke when discussing Table 4, where planning from scratch is able to solve the problem in the allotted memory and the modified strategy is not.

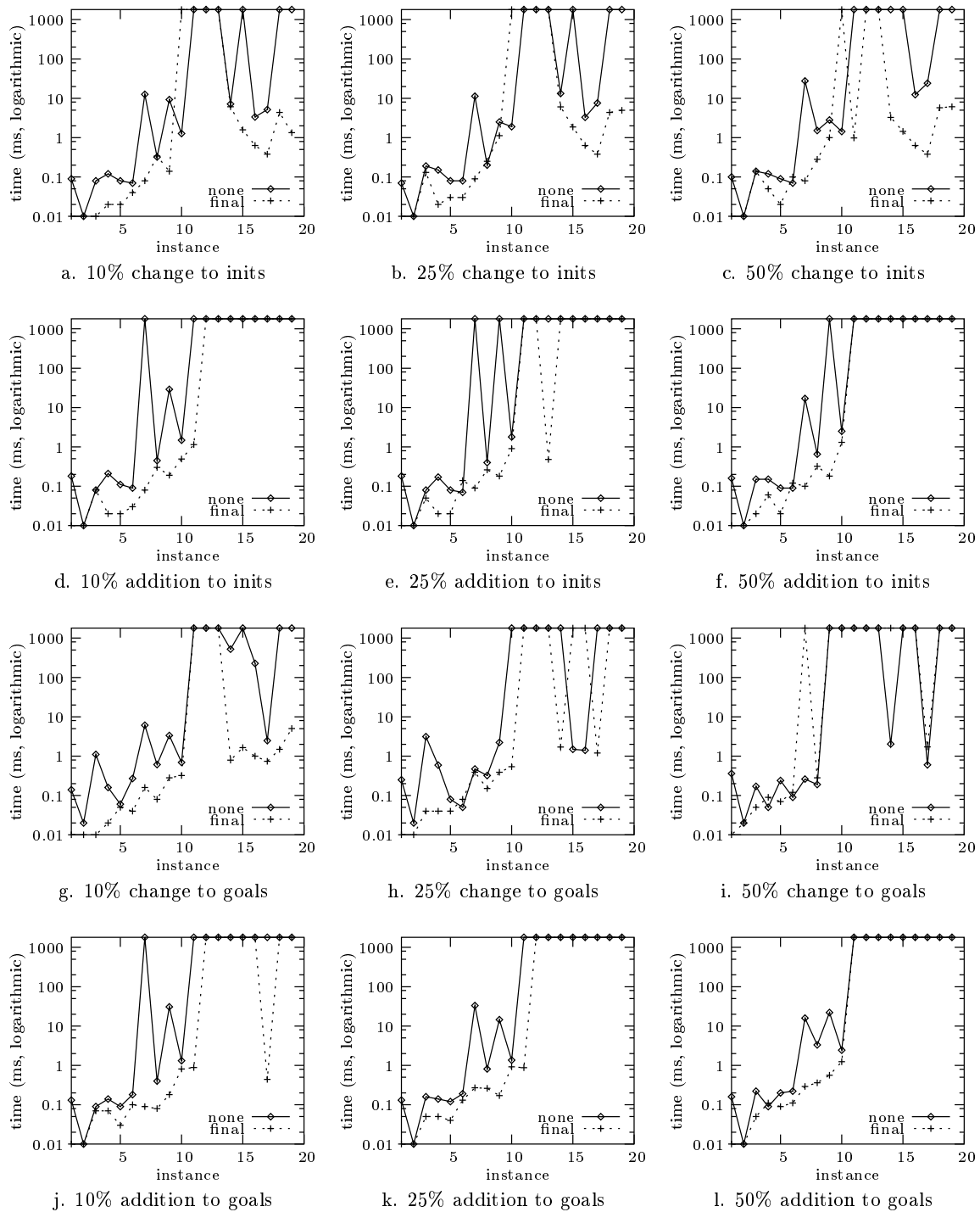


Figure 6: Runtimes of the *final* strategy compared with planning from scratch

Table 5: χ^2 test for the whole population of strategies for all types of changes (wins over none)

	percentage	all	all*	helpful	helpful*	supported	supported*	HS	HS*	final	final*	χ^2	p
I change	10%	8	10	8	10	12	7	12	7	13	13	5.2	0.82
	25%	7	8	7	9	13	5	13	7	13	12	8.9787	0.44
	50%	6	9	6	9	12	4	10	5	13	13	11.5057	0.24
I add	10%	16	15	16	14	16	14	16	13	16	16	0.7632	1.00
	25%	11	10	11	11	11	10	11	8	11	11	0.8095	1.00
	50%	3	4	3	3	4	3	3	3	3	4	0.6364	1.00
G change	10%	9	9	9	8	9	8	10	8	9	11	0.8889	1.00
	25%	7	6	8	6	7	5	7	7	9	8	1.7143	1.00
	50%	6	4	7	5	6	3	7	4	8	8	4.7586	0.85
G add	10%	10	8	11	7	10	7	10	6	11	10	3.3333	0.95
	25%	10	7	10	5	9	4	9	4	10	11	8.2152	0.51
	50%	7	7	8	6	7	3	7	3	8	8	5.0625	0.83

4.2.1 Hypothesis 3: Makespans

The second hypothesis concerned the quality of the solutions. When we introduced the strategies in Section 3, we already noted that some strategies might not work on all problems. Indeed, Figure 3 showed a number of problems that could not be solved at all. That still leaves us with the question: if the problem *is* solvable, is the quality of the solution affected?

First, let us get a feel for the data again, and consider Figure 7. This shows the quality of the solutions obtained on the *Logistics-A* set, using the *thin gp-based* encoding. Notice that planning from scratch results in the optimal makespan; so it is not possible for the strategies to do better than that. Just from looking at these graphs, we can already discern a pattern: except for the *final* and *final** strategies, all of the strategies perform worse on quite a number of instances. Surprisingly, this also holds for the relaxed strategies, for which we thought this would be less of an issue. We think that the reason for this is that the strategies sometimes enforce actions that do not make part of the optimal plan for the new situation. So even though we allow more freedom regarding the timesteps that these actions should occur, it still results in a less than optimal plan.

The full results are presented in Table 6. This table shows the number of times that a strategy produces a plan that was *worse* than the plan produced by planning from scratch.⁶ To compile this table, we only took into account those instances in which planning from scratch was able to generate a solution within the allotted time. The number of plans thus taken into account is given by the number in parenthesis behind the name of each encoding. What we can see is that all encodings produce results that are suboptimal. Also, in most cases the differences between the strategies are not significant. This means that we have to partly

⁶Remember that when it comes to size, a plan is considered better than another, if it has fewer actions.

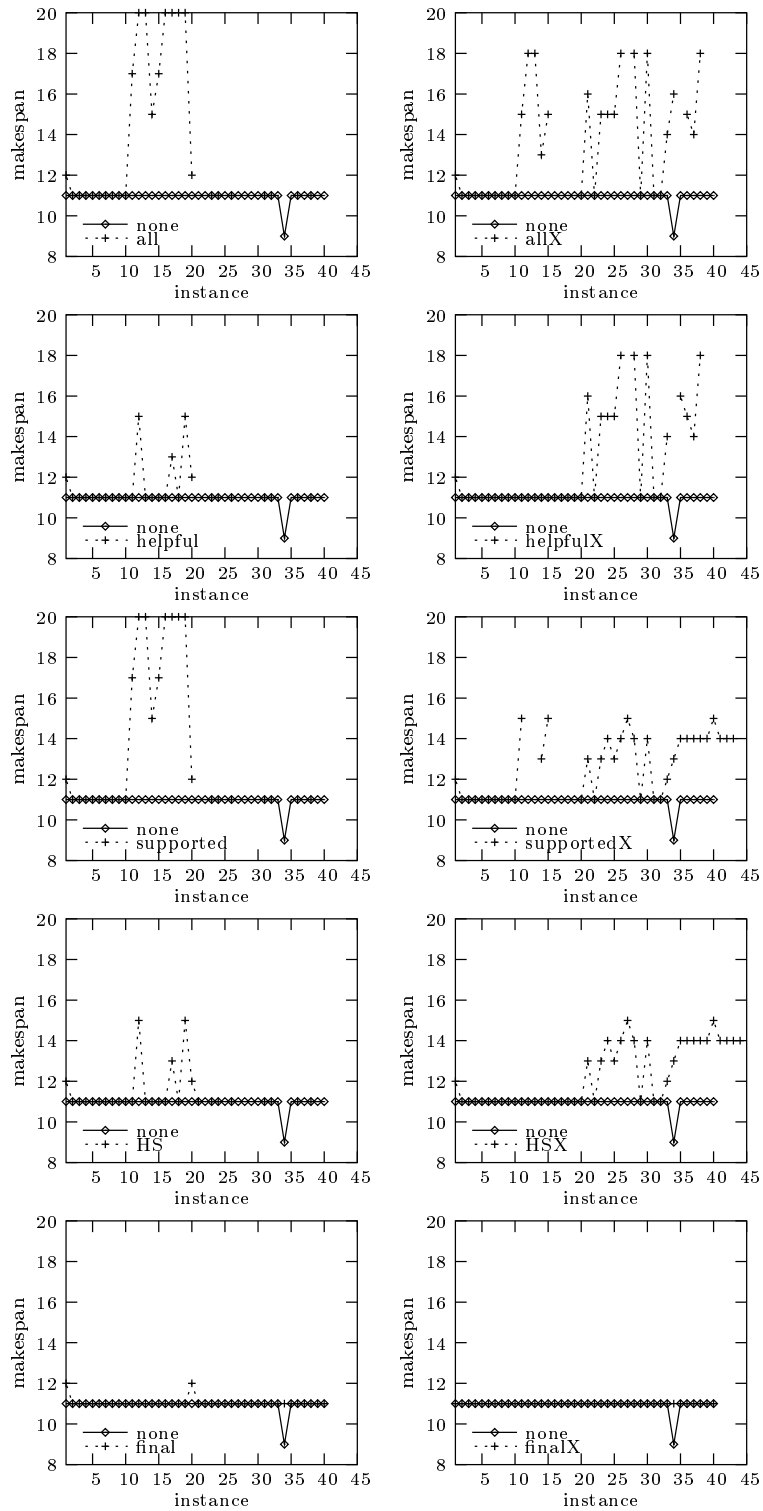


Figure 7: Quality of the solutions obtained for the various strategies. *Logistics-A* domain; *thin-gp-based* encoding.

Table 6: χ^2 test for the whole population of strategies (number of results *worse* than planning from scratch)

	encoding	all	all*	helpful	helpful*	supported	supported*	HS	HS*	final	final*	χ^2	p
logistics	action-based (85)	7	10	4	7	7	10	4	7	4	1	11.9508	0.22
	gp-style action-based (129)	32	35	28	32	32	35	28	32	28	23	4.082	0.91
	gp-based (129)	31	34	27	31	31	34	27	31	27	22	4.2203	0.90
	thin gp-based (129)	33	36	29	33	33	36	29	33	29	24	3.9524	0.91
rocket	action-based (90)	36	35	21	20	36	34	21	19	18	16	25.0938	< 0.01
	gp-style action-based (88)	37	40	22	25	37	38	22	23	19	18	24.6584	< 0.01
	gp-based (90)	37	40	22	25	37	38	22	23	19	18	24.6584	< 0.01
	thin gp-based (90)	37	40	22	25	37	38	22	23	19	18	24.6584	< 0.01
gripper	action-based (0)	0	0	0	0	0	0	0	0	0	0		
	gp-style action-based (0)	0	0	0	0	0	0	0	0	0	0		
	gp-based (2)	2	2	2	2	2	2	2	2	2	2	0	1.00
	thin gp-based (2)	2	2	2	2	2	2	2	2	2	2	0	1.00
satellite	action-based (38)	26	26	26	26	20	20	20	20	23	23	3.1304	0.96
	gp-style action-based (124)	84	82	84	80	67	64	64	61	70	62	11.5543	0.24
	gp-based (141)	98	96	97	93	81	76	78	73	77	69	13.0501	0.16
	thin gp-based (139)	98	96	98	94	81	76	78	73	77	69	13.5714	0.14

reject our second hypothesis: plan adaptation does lead to less efficient plans, but relaxing the strategies does not lead to an improvement in this regard. The reason for this has been given above: even though the actual time steps may be varied, the strategies all include some actions that force a sub optimal plan.

5 Conclusions

Compilation techniques have been very successfully applied to planning, yet have not been used for plan adaptation purposes. As we consider plan adaptation an important real-world issue, either for plan repair purposes or for the use in case-based planning, we investigated a SAT-based plan adaptation framework. One important condition that we set out with is that we want to use an off-line Satisfiability solver, due to the lack of general online methods for this problem. Therefore, our approach works as follows. Given the original planning problem Π_0 , its solution Δ_0 and a current problem Π , we use a SAT-based planner to generate a SAT-encoding for Π from scratch. Our system then combines the SAT-encoding with information from Π_0 and Δ_0 to produce a new SAT instance, that reflects some common aspects of Π and Π_0 , to help the solver in its job. This is then passed to the solver to obtain a solution.

The main result of this paper is that we have shown that such a framework is indeed possible. We investigated several possible modification strategies on top of SatPlan 2006, of which two in particular (*final* and *final**) worked really well. These strategies both achieve a significant speed-up over planning from scratch. However, these strategies still suffer from a reduction in plan quality, as do the other methods. This

is an issue that is more often encountered in plan adaptation: parts may be re-used that are actually not required in the new solution, leading to less efficient plans.

These results encourage future work. On the one hand, we are interested in extending the work to include planners based on other formalisms, such as Constraint Programming and Integer Programming. Especially constraint programming has our attention, as the emphasis in recent years has shifted to combinations of CP and planning.

On the other hand, we are interested in further exploring the framework presented here, based on Satisfiability. We observed a difference between the different encodings. A better understanding of these differences may lead to specialised encodings that are particularly well-suited to plan adaptation problems. In particular, we are interested in strategies that would suffer less from the reduction in plan quality. Also we are interested in adapting the actual SAT solvers themselves to take into account information from previous runs. This would be a generation of the clause learning performed by MaxSat that we discussed in Section 2: instead of doing this within different attempts at the same problem, we would like to apply it between attempts at different problems. Eventually, such work may even lead to general online SAT solvers.

References

- [1] Mitchell Ai-Chang, John Bresina, Kimberly Farrell, Jennifer Hsu, Ari Jónsson, Bob Kanefsky, Michael McCurdy, Paul Morris, Kanna Rajan, Alonso Vera, Jeffrey Yglesias, Leonard Charest, and Pierre Maldague. MAPGEN: Mixed-initiative activity planning for the mars exploration rover mission. In *Proceedings of Demonstration Systems Track, ICAPS-03*, 2003.
- [2] Peter Van Beek and Xinguang Chen. CPlan: A constraint programming approach to planning. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590, Menlo Park, CA, July 1999. AAAI Press.
- [3] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
- [4] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.
- [5] Yixin Chen, Zhao Xing, and Weixiong Zhang. Long-distance mutual exclusion for propositional planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1840–1845, Hyderabad, India, 2007.

- [6] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, San Francisco, CA, 1995.
- [7] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
- [8] Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-compilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1169–1177, Yokohama, Japan, 1997.
- [9] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: replanning versus plan repair. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-06)*, pages 193–202, UK, 2006.
- [10] M.R. Garey and D.S. Johnson. *Computers and intractability – a guide to the theory of NP-completeness*. W.H. Freeman and company, New York, NY, 1979.
- [11] A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proc. of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, pages 112–121, Menlo Park, CA, 2000. AAAI Press.
- [12] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solver. In *Design Automation and Test in Europe (DATE)*, pages 142–149, 2002.
- [13] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 431–437, Madison, Wisconsin, 1998.
- [14] J. Hoffmann and S. Edelkamp. The deterministic part of IPC-4: An overview. *JAIR*, 24:519–579, 2005.
- [15] Subbarao Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, 1997.
- [16] Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2):167–238, 1995.
- [17] Henry Kautz and B Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the workshop on planning as combinatorial search, held in conjunction with AIPS'98*, 1998.

- [18] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, Menlo Park, CA, 1996. AAAI Press.
- [19] Henry Kautz, Bart Selman, and Joerg Hoffmann. SatPlan: Planning as satisfiability. In *Abstracts of the 5th International Planning Competition*, 2006.
- [20] Henry A. Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR '96)*, pages 374–384, 1996.
- [21] D. Leake. *Case-Based Reasoning : Experiences, Lessons and Future Directions*. AAAI Press, Menlo Park, CA, 1996.
- [22] Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *Journal of AI Research*, 20:1–59, 2003.
- [23] Adriana Lopez and Fahiem Bacchus. Generalizing graphplan by formulating planning as a csp. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 954–960, 2003.
- [24] A. Nadel. The jerusat SAT solver. Master’s thesis, Hebrew University of Jerusalem, Jerusalem, Israel, 2002.
- [25] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers, San Mateo, CA, 2004.
- [26] Bernhard Nebel and Jana Koehler. Plan modifications versus plan generation: A complexity-theoretic perspective. Technical Report RR-92-48, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1992.
- [27] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12):1031–1080, 2006.
- [28] Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master’s thesis, Simon Fraser University, Vancouver, B.C., Canada, 2004.
- [29] Lucas Spalazzi. A survey on case-based planning. *Artificial Intelligence Review*, 16:3–36, 2001.
- [30] Sam Steel. The bread and butter of planning. *Artificial Intelligence Review*, 1:159–181, 1987.

- [31] Menkes van den Briel and Subbarao Kambhampati. Optiplan: Unifying ip-based and graph-based planning. *Journal of AI Research*, 24:919–931, 2005.
- [32] Menkes van den Briel and Subbarao Kambhampati. IPPlan: Planning as integer programming. In *Abstracts of the 5th International Planning Competition*, 2006.
- [33] Roman van der Krogt. *Plan Repair in Single-Agent and Multi-Agent Systems*. PhD thesis, Delft University of Technology, Delft, The Netherlands, 2005.
- [34] Roman van der Krogt and Mathijs de Weerd. Plan repair as an extension of planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-05)*, Monterey, California, 2005.
- [35] Vincent Vidal and Sébastien Tabary. The new version of CPT, an optimal temporal POCL planner based on constraint programming. In *Abstracts of the 5th International Planning Competition*, 2006.
- [36] Zhao Xing, Yixin Chen, and Weixiong Zhang. Optimal STRIPS planning by maximum satisfiability and accumulative learning. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS-06)*, pages 442–446, Cumbria, UK, 2006.

A Construction of the Satellite Problems

The Satellite domain was introduced in the 2002 International Planning Competition, and used again during the 2004 competition. The domain involves the planning of measurements with a number of satellites. Each satellite is equipped with a number of instruments, that can make images in different modes (e.g. infrared, or thermal). In order to use an instrument, it has to be calibrated first. This involves turning the satellite in the direction of the calibration target and taking a measurement. Then the satellite can turn in the required direction and take an image of the actual target.

The Satellite problems are all derived from the 20 STRIPS problems of the 2002 and 2004 IPC benchmark sets.⁷ These problems are of increasing size and complexity. Four types of changes were made to the 20 base problems, in three different degrees.

Changes to the Initial State The first type of derivations we made feature changes made to the initial state. In order to localise the changes to the initial state, three types of changes were allowed:

⁷A generator for the original problems can be downloaded from <http://planning.cis.strath.ac.uk/competition/CompoDomains/IPC3small.tgz>. The generator for the adapted problems can be found at <http://roman.vanderkrogt.org/software/satgen-adapt.tgz>.

1. the supported modes of the instruments;⁸
2. the calibration targets of the instruments; and
3. the initial direction of the satellite.

In order to decide which changes to make, we generate the list of propositions in the initial state specifying the modes, targets and directions. To make a single change, we randomly select one of those propositions and change it to a (different) random value. For example, if we select a proposition that specifies the calibration target of instrument 5, we change it so that instrument 5 now has a different calibration target. To generate the problems, we made changes to 10, 25 and 50 percent of the propositions that can be changed (with a minimum of 1).

Additions to Initial State These problems were derived from the base problems by adding propositions to the initial state. This enlarges the search space, as more satellites or instruments become available, but does not invalidate the original plan. It does tend to make the original plan less efficient, as an additional satellite may be used in parallel to the existing ones, resulting in a shorter makespan.

To decide whether to add an instrument to an existing satellite, or to generate a new satellite (including some instruments), we look at the ratio between instruments and satellites. If there are s satellites with a total of i instruments, we add a new instrument with probability $\frac{i}{s+i}$, and a new satellite with probability $\frac{s}{s+i}$. We add successive instruments or satellites until we have increased the number of propositions that can be changed (as used in the previous paragraph) by at least 10, 25 or 50 percent. (Adding a satellite adds a number of new propositions, and may push us over the number required.)

Changes to the Goals These problems were all found by changing a number of goals. Again, we want to localise the changes, and make sure that the new problem has a solution. The goals we change come in two categories:

1. the required mode of the requested images; and
2. the final direction of the satellites.

Similar to the changes made to the initial state, we collect the goal propositions and select a random one. If it specifies the mode for one of the requested images, we change it to another mode (that one of the available instruments is capable of). If it specifies the direction of a satellite, we change its final direction to another target. Again, we made changes to 10, 25 and 50 percent of the goals.

⁸We did not allow changes that would completely remove the capability of making images of a mode requested by the goals.

Additional Goals These problems were derived from the base problems by adding a number of goals. From our collection of goal propositions, we randomly select one. If it is a satellite, which has no final direction yet, we give it a final direction. If it is an image, we request an additional image of the same target, but with a different mode (and one that is not already a goal). It is possible that we select a satellite that already has a final direction, or an image whose target already is measured in all possible modes. In that case, we select a different proposition, until we find one that can be changed.

Some of the smaller problems have only requests for images. For this reason, we sometimes add a final direction to a satellite when we select an image proposition, and vice versa. For the test set used in this paper, we did so in 10% of the selected propositions.