Relaxation of Temporal Planning Problems

Martin C. Cooper, Frédéric Maris, Pierre Régnier

Relaxation of Temporal Planning Problems *

Martin C. Cooper Frédéric Maris Pierre Régnier
IRIT
University of Toulouse
Toulouse, France
{cooper, maris, regnier}@irit.fr

***Abstract*—Relaxation is ubiquitous in the practical resolution of combinatorial problems. If a valid relaxation of an instance has no solution then the original instance has no solution. A tractable relaxation can be built and solved in polynomial time. The most obvious application is the efficient detection of certain unsolvable instances. We review existing relaxation techniques in temporal planning and propose an alternative relaxation inspired by a tractable class of temporal planning problems. Our approach is orthogonal to relaxations based on the ignore-all-deletes approach used in non-temporal planning. We show that our relaxation can even be applied to non-temporal problems, and can also be used to extend a tractable class of temporal planning problems.**

Temporal planning, relaxation, monotonicity.

I. INTRODUCTION

Propositional non-temporal planning consists in finding a sequence of actions which transforms an initial state into a goal state. Each action can be executed only if a set of conditions is satisfied and the effect of its execution is to instantaneously change the truth values of a subset of the propositional variables describing the state of the world. It is well known that propositional planning is PSPACE-Complete [2]. In temporal planning, actions have a duration, and the moments at which conditions must hold or at which changes to the values of state variables occur are not necessarily simultaneous. Indeed, in the PDDL 2.1 temporal framework [17] [10], conditions can be imposed at the beginning, at the end or over the whole duration of an action, while effects can occur at the beginning or end of the action. In this framework, the PSPACE-complete complexity of classical planning can be preserved only when different instances of the same action cannot overlap; if they can overlap, testing the existence of a valid plan becomes an EXPSPACE-complete problem [18].

II. TEMPORAL PLANNING

We study temporal propositional planning in a language based on the temporal aspects of PDDL2.1. A *fluent* is a positive or negative atomic proposition. As in PDDL2.1, we consider that changes to the values of fluents are instantaneous but that conditions on the value of fluents may be imposed over an interval. An *action a* is a quadruple <Cond(a), Add(a), Del(a), Constr(a)>, where the set of conditions Cond(a) is the set of fluents which are required to be true for a to be executed, the set of additions Add(a) is the set of fluents which are established by a, the set of deletions Del(a) is the set of fluents which are destroyed by a, and the set of constraints Constr(a) is a set of constraints between the relative times of events which occur during the execution of a. An event corresponds to one of four possibilities: the establishment or destruction of a fluent by an action a, or the beginning or end of an interval over which a fluent is required by an action a. In PDDL2.1, events can only occur at the beginning or end of actions, but we extend this language so that events can occur at any time provided the constraints Constr(a) are satisfied. Note that Add(a) $\cap$ Del(a) may be non-empty. Indeed, it is not unusual for a durative action to establish a fluent at beginning of the action and destroy it at its end. We can also observe that the duration of an action, the time between the first and last events of the action, does not need to be explicitly stored.

We use the notation $a \rightarrow f$ to denote the event that action a establishes fluent f, $a \rightarrow \neg f$ to denote the event that a destroys f, and $f \mid\!\rightarrow a$ and $f \rightarrow\!\mid a$, respectively, to denote the beginning and end of the interval over which a requires the condition f. If f is already true (respectively, false) when the event $a \rightarrow f$ ($a \rightarrow \neg f$) occurs, we still consider that a establishes (destroys) f. We use the notation $\tau(E)$ to represent the time in a plan at which an event E occurs. For a given action a, let Events(a) represent the different events which constitute its definition, namely $(a \rightarrow f)$ for all f in Add(a), $(a \rightarrow \neg f)$ for all f in Del(a), $(f \mid\!\rightarrow a)$ and $(f \rightarrow\!\mid a)$ for all f in Cond(a). The definition of an action a includes constraints Constr(a) on the relative times of events in Events(a). As in PDDL2.1, we consider that the length of time between events in Events(a) is not necessarily fixed and that Constr(a) is a set of interval constraints on pairs of events, such as $\tau(f \rightarrow\!\mid a) - \tau(f \mid\!\rightarrow a) \in [\alpha, \beta]$ for some constants α,β. We use $[\alpha_a(E_1,E_2), \beta_a(E_1,E_2)]$ to denote the interval of possible values for the relative distance between events E_1,E_2 in action a. A fixed length of time between events E_1,E_2 can be modeled by setting $\alpha_a(E_1,E_2) = \beta_a(E_1,E_2)$ and open-ended intervals by setting $\alpha_a(E_1,E_2) = -\infty$ or $\beta_a(E_1,E_2) = \infty$. We now introduce two basic constraints that all temporal plans must satisfy. In general a plan may contain multiple instances of the same action which we can represent by a multi-set A.

inherent constraints: $\forall a \in A$, a satisfies Constr(a), i.e. $\forall E_1,E_2 \in$ Events(a), $\tau(E_1) - \tau(E_2) \in [\alpha_a(E_1,E_2), \beta_a(E_1,E_2)]$.

* This work is supported by ANR Project ANR-10-BLAN-0210.

contradictory-effects constraints: $\forall a_i,a_j \in A$, for all positive fluents $f \in \text{Del}(a_i) \cap \text{Add}(a_j)$, $\tau(a_i \rightarrow \neg f) \neq \tau(a_j \rightarrow f)$.

Definition 1. A *temporal planning problem* $\langle I,A,G \rangle$ consists of a set of actions A, an initial state I and a goal G, where I and G are sets of fluents.

Definition 2. $P = \langle A',\tau \rangle$, where A' is a multi-set of actions $\{a_1,...,a_n\}$ and τ is a real-valued function on Events(A') (the union of the multisets Events(a) for $a \in A'$), is a *(temporal) plan* for the problem $\langle I,A,G \rangle$ if
(1) each element of A' is an instance of an action in A,
(2) P satisfies the inherent and contradictory-effect constraints on A'; and
when P is executed (i.e. fluents are established or destroyed at the times given by τ) starting from the initial state I:
(3) $\forall a_i \in A'$, each $f \in \text{Cond}(a_i)$ is true when it is required,
(4) all goals $g \in G$ are true at the end of the execution of P.
(5) P is robust under infinitesimal shifts in the starting times of actions.

A plan P is *minimal* if no subset of P is a valid plan.

Condition (5) means that we disallow plans which require perfect synchronization between different actions. This condition can be imposed within PDDL2.1 [11]. We require that in all plans fluents are established *strictly* before the beginning of the interval over which they are required. The only exception to this rule is when a fluent f is established and required by the same action a. We allow the possibility of perfect synchronization within an action, which means that we can have $\tau(a \rightarrow f) = \tau(f \mid\!\rightarrow a)$. Similarly, fluents can only be destroyed *strictly* after the end of the interval over which they are required. The only exception to this rule is when a fluent f is required and destroyed by an action a, in which case we can have $\tau(f \rightarrow\!\mid a) = \tau(a \rightarrow \neg f)$.

A temporal planning problem $\langle I,A,G \rangle$ is *positive* if there are no negative fluents in the conditions of actions nor in the goal G. It is well known that any planning problem can be transformed into an equivalent positive problem in linear time [13]. Thus, in this paper, we only consider positive temporal planning problems $\langle I,A,G \rangle$. By this assumption, G and Cond(a) (for any action a) are composed of positive fluents. By convention, Add(a) and Del(a) are also composed exclusively of positive fluents. The initial state I, however, may contain negative fluents.

III. EU Monotone Planning

In this section, we introduce the notions of establisher-uniqueness and monotonicity of fluents. Together, these two conditions are sufficient for the existence of a polynomial-time algorithm for temporal planning [6]. Establisher-uniqueness is similar to post-uniqueness in SAS^+ planning [14] restricted to Boolean variables.

Definition 3. A set of actions $A=\{a_1,...,a_n\}$ is *establisher-unique* (EU) relative to a set of positive fluents S if for all $i \neq j$, $\text{Add}(a_i) \cap \text{Add}(a_j) \cap S = \varnothing$, i.e. no fluent of S can be established by two distinct actions of A.

If a set of actions is EU relative to the set of sub-goals (recursively defined as the minimum set of fluents which belong to G or to the conditions of some action which establishes a sub-goal) of a problem, then we can determine in polynomial time a set of actions which are necessarily present in a temporal plan. In general, other actions may be required to re-establish fluents which were present in I but have been destroyed by another action. There also remains the problem of determining how many times each action must occur and then scheduling these action-instances in order to produce a valid temporal plan. These problems can be solved in polynomial time if we also impose monotonicity of fluents [6].

Definition 4. A fluent f is *–monotone* if, after being destroyed f is never re-established in any temporal plan. A fluent f is *+monotone* if, after having been established f is never destroyed in any temporal plan. A fluent is *monotone* if it is either + or –monotone.

Example 1: In fairly obvious contexts, the fluents alive or brand-new are –monotone, whereas the fluents dissolved, cooked, graduated, born and extinct are all +monotone.

If A is a set of actions, we use the notation Del(A) to represent the union of the sets Del(a) ($\forall\ a \in A$). Add(A), Cond(A), Constr(A) are defined similarly. The following lemma follows trivially from Definition 4.

Lemma 1. If $f \notin \text{Add}(A) \cap \text{Del}(A)$, then f is both –monotone and +monotone.

We now introduce three other types of constraints. The –authorisation (resp, +authorisation) constraint is applied only to –monotone (resp, +monotone) positive fluents f, whereas the causality constraint is applied to all monotone fluents.

–authorisation constraints on the positive fluent f: for all $a_i \neq a_j \in A$, if $f \in \text{Del}(a_j) \cap \text{Cond}(a_i)$, then $\tau(f \rightarrow\!\mid a_i) < \tau(a_j \rightarrow \neg f)$; for all $a_i \in A$, if $f \in \text{Del}(a_i) \cap \text{Cond}(a_i)$, then $\tau(f \rightarrow\!\mid a_i) \leq \tau(a_i \rightarrow \neg f)$.

+authorisation constraints on the fluent f: $\forall a_i,a_j \in A$, if $f \in \text{Del}(a_j) \cap \text{Add}(a_i)$, then $\tau(a_j \rightarrow \neg f) < \tau(a_i \rightarrow f)$.

causality constraints on the positive fluent f: for all $a_i \neq a_j \in A$, if $f \in (\text{Cond}(a_j) \cap \text{Add}(a_i)) \backslash I$, then $\tau(a_i \rightarrow f) < \tau(f \mid\!\rightarrow a_j)$; for all $a_i \in A$, if $f \in (\text{Cond}(a_i) \cap \text{Add}(a_i)) \backslash I$ then $\tau(a_i \rightarrow f) \leq \tau(f \mid\!\rightarrow a_i)$.

If A is EU relative to the set of sub-goals, all sub-goals are monotone and all sub-goals in I are –monotone, then the temporal planning problem $\langle I,A,G \rangle$ is equivalent to solving the $STP^{\neq}$ (Simple Temporal Problem with disequality constraints) composed of the inherent, contradictory-effect, authorisation and causality constraints [6], and can hence be

solved in polynomial time [15]. It is clearly polynomial-time to detect whether all actions are EU. On the other hand, the very general definition of monotonicity of fluents implies that this is not the case for determining whether fluents are monotone. Indeed, determining whether a fluent is monotone is PSPACE-hard if overlapping instances of the same action are not allowed in plans and EXPSPACE-complete otherwise [6]. We will return to the detection of monotonicity later in this paper. However, this is not an issue for the definition of a relaxation, since it is relatively easy to construct a relaxed instance in which all fluents are monotone. For example, Lemma 1 tells us that eliminating f from Del(a) for all a renders f monotone. In the next section we describe a stronger form of relaxation which allows us to retain the destruction of fluents.

IV. TEMPORAL RELAXATION

We first show that the standard form of relaxation used in propositional planning, consisting of simply ignoring all destructions of fluents and then trying to attain the goals by successively applying all relaxed actions whose conditions are satisfied [1], does not directly generalize to temporal planning. An important aspect of temporal planning, which is absent from non-temporal planning, is that certain temporal planning problems, known as temporally-expressive problems, require concurrency of actions in order to be solved [7]. A typical example of a temporally-expressive problem is cooking: several ingredients must be cooked simultaneously in order to be ready at the same moment. A subclass of temporally expressive problems, known as temporally-cyclic, require cyclically-dependent sets of actions in order to be solved [5].

A simple example of a temporally-cyclic problem is the building of two pieces of software by two different subcontractors, each needing to know the specification of the other program in order to complete their own program by building the interface with the other program. We can model this by two durative actions, action A_1 (A_2) having $spec_1$ ($spec_2$) as an effect at its beginning and $spec_2$ ($spec_1$) as a condition at its end. It is easy to see that neither action can occur in a plan without the other and that they must overlap. The standard form of relaxation, as described above, would not be able to accept either of the two actions since it tries one action at a time. Thus, certain proposed relaxations, although very useful in guiding heuristic search [9] [8], do not produce a valid relaxation for temporally cyclic problems. Different solutions exist to get round the problem of temporal cycles. For example, there is a polynomial-time algorithm to transform a temporally-cyclic problem into an equivalent acyclic one [5]. Other transformations have been proposed in the literature [16] [3] which also eliminate the possibility of temporal cycles, although this was not an explicitly-stated aim in the descriptions of these transformations: temporal cycles are avoided by decomposing durative actions into instantaneous actions denoting the start and end of the action. Intermediate conditions can also be managed by splitting actions into component actions enclosed within an "envelope" action [19]. In each case, ignoring deletes in the transformed problem is a valid relaxation.

In this section, we present an alternative form of relaxation, inspired by EU monotone planning, consisting of an $STP^{\neq}$ instance which has a solution if the original temporal planning instance has a solution.

By applying the following simple rule until convergence we can transform (in polynomial time) any temporal planning problem P into a relaxed version P' which is EU relative to the set of sub-goals S: if a sub-goal fluent f is established by two distinct actions, then delete f from the goal G and from Cond(a) for all actions a. As a consequence, f is no longer a sub-goal. Clearly, P' is a valid relaxation of P. From now on we assume the temporal planning instance is EU relative to S.

We denote by A^{ind} the set of actions which have been detected as indispensable in all plans [4]. Establisher-uniqueness implies that we can easily identify many such actions, in particular actions which establish sub-goals not present in the initial state I [6].

We cannot assume in the $STP^{\neq}$, which we call TR (for Temporal Relaxation), that a single instance of each action will be sufficient. For each indispensable action a and for each event $E \in$ Events(a), we introduce two variables $\tau_{\text{first}}(E)$, $\tau_{\text{last}}(E)$ representing the times of the first and last occurrences of event E in the plan. The constraints of TR include versions of the internal, contradictory-effects, authorization and causality constraints (which we give below) together with the obvious

intrinsic constraint: $\forall a \in A^{\text{ind}}$, for all events $E \in$ Events(a), $\tau_{\text{first}}(E) \leq \tau_{\text{last}}(E)$.

We make the assumption that no two instances of the same action can overlap, to conserve the PSPACE complexity of classical planning [18]. Under this assumption, we know that for $E_1,E_2 \in$ Events(a), the first occurrences of E_1,E_2 in a plan correspond to the same instance of action a. A similar remark holds for the last occurrences of E_1,E_2. This means that we can apply in TR each inherent constraint in Constr(a) independently to the values of $\tau_{\text{first}}(E)$ and $\tau_{\text{last}}(E)$ ($E \in$ Events(a)).

inherent constraint: $\forall a \in A^{\text{ind}}$, $\forall\ E_1,E_2 \in$ Events(a),
$\tau_{\text{first}}(E_1) - \tau_{\text{first}}(E_2) \in [\alpha_a(E_1,E_2), \beta_a(E_1,E_2)]$ and
$\tau_{\text{last}}(E_1) - \tau_{\text{last}}(E_2) \in [\alpha_a(E_1,E_2), \beta_a(E_1,E_2)]$.

The contradictory-effects constraints in TR are as follows:
contradictory-effects constraints: $\forall a_i,a_j \in A^{\text{ind}}$, for all positive fluents $f \in \text{Del}(a_i) \cap \text{Add}(a_j)$, $\tau_{L1}(a_i \rightarrow \neg f) \neq \tau_{L2}(a_j \rightarrow f)$, $\forall L1,L2 \in \{\text{first,last}\}$.

For each positive fluent f which is known to be $\neg$monotone (resp, +monotone), we apply in TR the

following modified version of the –authorisation constraints (resp, +authorisation constraints):

–authorisation constraints on f: $\forall\ a_i \neq a_j \in A^{ind}$, if $f \in \text{Del}(a_j) \cap \text{Cond}(a_i)$, then $\tau_{last}(f \rightarrow\mid a_i) < \tau_{first}(a_j \rightarrow \neg f)$; for all $a_i \in A^{ind}$, if $f \in \text{Del}(a_i) \cap \text{Cond}(a_i)$, then $\tau_{last}(f \rightarrow\mid a_i) \leq \tau_{first}(a_i \rightarrow \neg f)$.

+authorisation constraints on f: $\forall a_i, a_j \in A^{ind}$, if $f \in \text{Del}(a_j) \cap \text{Add}(a_i)$, then $\tau_{last}(a_j \rightarrow \neg f) < \tau_{first}(a_i \rightarrow f)$.

We check that every condition and every goal can be established, i.e. $\text{Cond}(A^{ind}) \subseteq I \cup \text{Add}(A)$ and $G \subseteq I \cup \text{Del}(A^{ind}) \cup \text{Add}(A)$. If not, we consider that the relaxation TR has no solution. We also apply in TR the following causality constraints for each positive fluent f.

causality constraints: $\forall\ a_i \neq a_j \in A^{ind}$, if $f \in (\text{Cond}(a_j) \cap \text{Add}(a_i)) \backslash I$ then $\tau_{first}(a_i \rightarrow f) < \tau_{first}(f \mid\rightarrow a_j)$; for all $a_i \in A^{ind}$, if $f \in (\text{Cond}(a_i) \cap \text{Add}(a_i)) \backslash I$ then $\tau_{first}(a_i \rightarrow f) \leq \tau_{first}(f \mid\rightarrow a_i)$.

We also apply the following goal constraints for each $g \in G$.

goal constraints: $\forall a_i, a_j \in A^{ind}$, if $g \in \text{Del}(a_j) \cap \text{Add}(a_i)$, then $\tau_{last}(a_j \rightarrow \neg g) < \tau_{last}(a_i \rightarrow g)$.

Of course, the causality and goal constraints are necessary conditions for the existence of a plan only if the temporal planning instance is EU relative to $(\text{Cond}(A^{ind}) \backslash I) \cup (G \cap \text{Del}(A))$. But this follows from the fact that we assume that the instance is EU relative to the set of all sub-goals.

If for an action $a \in A^{ind}$, all fluents in $\text{Add}(a)$ are known to be monotone, then only one instance of a occurs in minimal plans [6]; hence, for each event $E \in \text{Events}(a)$, we replace the two variables $\tau_{first}(E)$, $\tau_{last}(E)$ in the above constraints by a unique variable $\tau(E)$.

Clearly, TR is a valid relaxation since the constraints of TR must be satisfied by any valid plan. We state this formally in the form of a proposition.

Proposition 1. Let $\langle I,A,G\rangle$ be a temporal planning problem which is EU relative to its set of sub-goals S. Let $A^{ind} \subseteq A$ be a set of indispensable actions, i.e. actions which necessarily occur in all plans. If the temporal relaxation TR has no solution, then the temporal planning problem $\langle I,A,G\rangle$ has no solution.

Under assumptions of establisher-uniqueness and monotonicity, TR is in fact a solution procedure for the tractable class described by Cooper et al. [6].

Example 2: We now show that, even in non-temporal propositional planning, the temporal relaxation TR can detect unsolvable problems. In this example, all actions are instantaneous and hence we present it in the form of a non temporal planning problem P with initial state I={j,m,d}, goal G={g} and the following three actions:

Buy: j, m → h, ¬d, ¬m
Sell: h → m, ¬h
Mort2: d, h → m, ¬d, g

We can interpret the fluents as follows: j = I have a job, m = I have money, d = I am debt-free, h = I own a house, g = I have taken out a second mortgage. For example, the action Buy is possible only if I have a job and money to put down a deposit on a house; the result is that I own a house but I am in debt and no longer have money. The goal is to take out a second mortgage via the action Mort2.

This problem has no solution, but this fact is not detected by the standard relaxation of non-temporal planning problems consisting of ignoring all destructions of fluents. To set up TR, we first determine the indispensable actions A^{ind}= {Buy, Mort2} easily identified as indispensable by the rules given by Cooper et al. [4] since they establish the sub-goals h and g, respectively, not present in the initial state. Observe that A^{ind} is EU relative to the set of sub-goals. The STP$^{\neq}$ TR contains the constraints: $\tau_{first}(\text{d} \rightarrow\mid \text{Mort2}) < \tau_{last}(\text{d} \rightarrow\mid \text{Mort2})$ and $\tau_{first}(\text{d} \rightarrow\mid \text{Mort2}) = \tau_{first}(\text{h} \mid\rightarrow \text{Mort2})$ by intrinsic and internal constraints in Mort2; $\tau_{first}(\text{Buy} \rightarrow \text{h}) = \tau_{first}(\text{Buy} \rightarrow \neg\text{d})$ by an internal constraint in Buy; $\tau_{first}(\text{Buy} \rightarrow \text{h}) < \tau_{first}(\text{h} \mid\rightarrow \text{Mort2})$ by the causality constraint on h; $\tau_{last}(\text{d} \rightarrow\mid \text{Mort2}) < \tau_{first}(\text{Buy} \rightarrow \neg\text{d})$ by the –authorisation constraint, since d is –monotone (by Lemma 1). This set of five constraints has no solution, from which we can deduce that P has no solution. This example shows that temporal relaxation can be useful even in non-temporal planning problems.

Example 3: We now give a generic example involving the choice between two alternatives in which the temporal relaxation TR can detect unsolvable problems that cannot be detected by ignoring deletes. This simple example consists of a non-temporal planning problem with initial state I={f}, goal G={g,h} and the following two actions:

B: f → ¬f, g
C: f → ¬f, h

The fluents have many possible interpretations, including: f = I have a packet, g = I have sent the packet to Destination1, h = I have sent the packet to Destination2. Clearly this problem has no solution, but this is not discovered by the ignoring-deletes relaxation (which cannot take into account the fact that I no longer have the packet once I have sent it somewhere).

On the other hand, TR detects unsolvability as follows. Firstly, note that the problem is establisher-unique, both actions are indispensable (since they both establish fluents in $G \backslash I$) and all fluents are both + and –monotone by Lemma 1. Since all fluents in Add(B) and Add(C) are monotone, TR has a single variable $\tau(E)$ for each event E. Since f is –monotone, TR contains the two authorisation constraints: $\tau(\text{f} \rightarrow \text{C}) < \tau(\text{B} \rightarrow \neg\text{f})$ and $\tau(\text{f} \rightarrow \text{B}) < \tau(\text{C} \rightarrow \neg\text{f})$. TR also contains the inherent constraints $\tau(\text{f} \rightarrow \text{B}) = \tau(\text{B} \rightarrow \neg\text{f})$ and

$\tau(f \rightarrow C) = \tau(C \rightarrow \neg f)$, which immediately leads to a contradiction.

The above examples show that the EU monotone relaxation TR can be stronger than any relaxation based on ignoring deletes. To see that ignoring deletes can be stronger than EU monotone relaxation, consider a problem in which the unique goal g is produced by a unique action a such that Cond(a) = {f} where the fluent f is produced by two distinct actions b and c. In the EU monotone relaxation, the fluent f is deleted from Cond(a), since it is established by two distinct actions, and the relaxed version of the problem is immediately solvable by a plan containing the single action a. Ignoring deletes, on the other hand, can detect the unsolvability of the original problem in certain cases, for example, if Cond(b) and Cond(c) both contain fluents that are not in $I \cup$ Add(A).

An obvious application of temporal relaxation is the detection of indispensable actions [4]. Let P*[-a]* represent the planning problem P without a particular action a. If the temporal relaxation of P*[-a]* has no solution, then we can conclude that a is an indispensable action for P.

In the following sections we investigate other applications of temporal relaxation concerning the detection of different forms of monotonicity. The basic idea is that if H is a hypothesis to be tested and H can be expressed as the conjunction of STP$^{\neq}$ constraints, then we can add H to the constraints of the temporal relaxation TR to obtain an STP$^{\neq}$ instance TR[H]: if TR[H] has no solution then H cannot be true in any solution to the planning problem. In each case, the complexity of solving TR[H] is $O(n^3)$ time and $O(n^2)$ space, where n is the total number of events in the actions in A. This follows almost directly from the fact that the set of authorisation, inherent, contradictory-effects and causality constraints are STP$^{\neq}$ [15]. An instance of STP$^{\neq}$ can be solved in $O(n^3+k)$ time and $O(n^2+k)$ space [12], where n is the number of variables and k the number of inequations (i.e. constraints of the form $x_j - x_i \neq d$). Here, the only inequations are the contradictory-effects constraints of which there are at most n^2, so k=$O(n^2)$.

V. Detecting Monotonicity of Fluents

The detection of the monotonicity of fluents is essential for the *recognition* of instances of the polytime-solvable class of temporal planning problems described by Cooper at al. [6]. To detect the +monotonicity of a fluent f it suffices to give a proof that f cannot be destroyed in a plan after being established. Rules to provide such a proof, based on knowledge of the monotonicity of another fluent were given by Cooper et al. [6]. In this section, we give a more general proof rule which involves solving an STP$^{\neq}$ for each pair of actions a,b such that $f \in$ Add(a) $\cap$ Del(b). To try to prove that b cannot destroy f after a establishes f, we set up a relaxation TR[Before(a,f,b)] consisting of the temporal relaxation TR of the planning problem together with a single hypothesis constraint: Before(a,f,b) = $\{\tau_{first}(a \rightarrow f) < \tau_{last}(b \rightarrow \neg f)\}$.

To detect the –monotonicity of a fluent f we need to prove that f cannot be established in a plan after being destroyed. In the corresponding STP$^{\neq}$ TR[After(a,f,b)], the hypothesis is: After(a,f,b) = $\{\tau_{first}(b \rightarrow \neg f) < \tau_{last}(a \rightarrow f)\}$.

Lemma 2. Suppose that the set of actions A is EU. If TR[Before(a,f,b)] has no solution for any pair of actions a,$b \in A$ such that $f \in$ Add(a) $\cap$ Del(b), then f is +monotone. If TR[After(a,f,b)] has no solution for any pair of actions a,$b \in$ A such that $f \in$ Add(a) $\cap$ Del(b), then f is –monotone.

We can extend the polytime-solvable class of temporal planning problems described by Cooper et al. [6], using temporal relaxation to detect monotonicity of fluents. This new bigger class Π is still polytime-solvable. Each temporal relaxation can be solved in $O(n^3)$ time and $O(n^2)$ space, where n is the total number of events in the actions in A. The number of temporal relaxations to solve, in order to prove that a temporal planning problem belongs to Π, is proportional to the number of triples (a,f,b) such that a,$b \in A$ and $f \in$ Add(a) $\cap$ Del(b). The number of pairs (f,b) such that $b \in A$ and $f \in$ Del(b) is bounded above by n. If A is establisher-unique, then there is at most one action that $a \in A$ such that $f \in$ Add(a). Therefore, the complexity of recognizing Π is $O(n^4)$ time and $O(n^2)$ space. This can be compared with the $O(n^2)$ time and $O(n)$ space complexity to recognize the subclass of Π defined by simple rules for the recognition of monotonicity based on knowledge of the monotonicity of only one other fluent [6].

VI. Extending Monotonicity

In this section we introduce notions which extend the notion of monotonicity by considering only minimal plans, thus allowing us to define a larger tractable class of temporal planning than the class Π described in the previous section.

Definition 5. A fluent f is *–monotone** if, after being destroyed f is never re-established in any minimal temporal plan. A fluent f is *+monotone** if, after having been established f is never destroyed in any minimal temporal plan. A fluent is *monotone** if it is either + or –monotone*.

Example 4. To give an example of a monotone* fluent which is not monotone, consider the following planning problem in which all actions are instantaneous:

Start_vehicle: $e \rightarrow f$
Drive: $f \rightarrow g, \neg f$
Unload: $g \rightarrow h$

with I = {e}, G = {h}. The fluents represent that I have the ignition key (e), the engine is on (f), the destination has been reached (g) and that the package has been delivered (h). There is only one minimal plan, namely Start_vehicle, Drive, Unload, but there is also the non-minimal plan Start_vehicle, Drive, Start_vehicle, Unload in which the fluent f is established, destroyed and then re-established. Hence f is –monotone* but not –monotone.

We make the assumption in the remainder of this section that no two instances of the same action can overlap in a plan. We cannot hope to detect all monotone* fluents in polynomial time since the detection of monotonicity itself is PSPACE-complete [6]. However, we will show that many monotone* fluents can be detected in polynomial time. We begin with a simple lemma to detect certain +monotone* fluents.

Lemma 3. If A is EU, then for all $a \in A$ such that $\text{Add}(a) \cap \text{Cond(A)} = \emptyset$, all $f \in \text{Add}(a) \cap (G \backslash I)$ are +monotone*.

Proof: Any plan must contain an instance of action a since a establishes a goal $f \in G \backslash I$ and A is EU. Since $\text{Add}(a) \cap \text{Cond(A)} = \emptyset$, all instances of a except the last can be deleted without affecting the validity of a plan. Thus a minimal plan contains exactly one instance of a. Furthermore, no fluent $f \in \text{Add}(a) \cap (G \backslash I)$ can be destroyed in a plan after being established by this last instance of a. Hence f is +monotone*.

We say that an action-instance a *usefully produces* a fluent h during the execution of a plan if h was false just before being established by a. We say that a *usefully produces the required fluent* h if a usefully produces h and either $h \in G$ or the fluent h is the condition of some action c in the plan such that $\tau(a \rightarrow h) < \tau(h \mid\rightarrow c)$. We can now state the following general proposition.

Proposition 2. Suppose that the set of actions A is EU relative to the set of sub-goals and let $a \in A$ be the unique action that establishes sub-goal f. **(a)** If $\forall b \in A$ such that $f \in \text{Del}(b)$, there is no minimal plan in which the last instance of b destroys f after a establishes f, and such that these instances of a and b usefully produce required fluents, then f is +monotone*. **(b)** If $\forall b \in$ A such that $f \in \text{Del}(b)$, there is no minimal plan in which the last instance of a establishes f after b destroys f, and such that these instances of a and b usefully produce required fluents, then f is –monotone*.

Proof: (a) Let P be a minimal plan in which the last instance of b destroys f after a establishes f. Then, by the hypothesis of the proposition, either the last instance of b in P or the first instance of a in P does not usefully produce a required fluent. Hence P cannot be minimal, since we could delete the last instance of b or the first instance of a from P to leave another valid plan. This contradiction shows that f is +monotone*. The proof of case **(b)** is similar.

We now give a lemma which allows us to deduce one of the hypotheses of Proposition 2 and hence to deduce that a fluent f is +monotone* or that it is –monotone*. To simplify the expression of the lemma, we suppose that there is a goal-achieving action a_G that must be executed at the end of all plans and such that $\text{Cond}(a_G)$=G. This simply means that goal fluents h do not need to be treated as special cases.

Lemma 4. Suppose that A is EU relative to the set of sub-goals S and let $a \in A$ be the unique action that establishes fluent $f \in S$. Let $b \in A$ be such that $f \in \text{Del}(b)$.

(a) Let $h \in S \cap \text{Add}(a)$ and $h' \in S \cap \text{Add}(b)$. If any of the following conditions hold, then there is no minimal plan P in which the last instance of b usefully produces the required fluent h' and destroys f after the first instance of a usefully produces the required fluent h and establishes f:
(1) either one of h, h' belongs to I and is –monotone*.
(2) for all actions c,c' such that $h \in \text{Cond}(c)$, $h' \in \text{Cond}(c')$, $\text{TR}[\text{Before}(a,f,b) \cup \text{For}(a,\text{first},h,c) \cup \text{For}(b,\text{last},h',c')]$ has no solution, where $\text{For}(x,\text{L},h,c) = \{\tau_{\text{L}}(x \rightarrow h) < \tau_{\text{last}}(h \mid\rightarrow c)\}$.
(3) h' is monotone* and for all actions c,c' such that $h \in \text{Cond}(c)$ and $h' \in \text{Cond}(c')$, $\text{TR}[\text{Before}(a,f,b) \cup \text{Once}(b) \cup \text{For}(a,\text{first},h,c) \cup \text{For}(b,\text{last},h',c')]$ has no solution, where $\text{Once}(x) = \{\tau_{\text{first}}(E) = \tau_{\text{last}}(E) \mid \text{E} \in \text{Events}(x)\}$.
(b) Let $h \in S \cap \text{Add}(a)$ and $h' \in S \cap \text{Add}(b)$. If any of the following conditions hold, then there is no minimal plan P in which the last instance of a usefully produces the required fluent h and establishes f after the first instance of b usefully produces the required fluent h' and destroys f:
(1) either one of h, h' belongs to I and is –monotone*.
(2) for all actions c,c' such that $h \in \text{Cond}(c)$, $h' \in \text{Cond}(c')$, $\text{TR}[\text{After}(a,f,b) \cup \text{For}(a,\text{last},h,c) \cup \text{For}(b,\text{first},h',c')]$ has no solution.
(3) h is monotone* and for all actions c,c' such that $h \in \text{Cond}(c)$ and $h' \in \text{Cond}(c')$, $\text{TR}[\text{After}(a,f,b) \cup \text{Once}(a) \cup \text{For}(a,\text{last},h,c) \cup \text{For}(b,\text{first},h',c')]$ has no solution.

Proof: (a) We suppose that A is EU relative to S, $f \in S \cap \text{Add}(a) \cap \text{Del}(b)$, $h \in S \cap \text{Add}(a)$ and $h' \in S \cap \text{Add}(b)$. Suppose that in a minimal plan P the last instance of the action b destroys f after f is established by a, that the first instance of a usefully produces the required fluent h and the last instance of b usefully produces the required fluent h'. We will show, in each case, that there is a contradiction.
(1) If $h \in I$ and h is –monotone*, then by the definition of –monotone*, no action can usefully produce h in P. A similar argument holds for h'.
(2) If $\text{TR}[\text{Before}(a,f,b) \cup \text{For}(a,\text{first},h,c) \cup \text{For}(b,\text{last},h',c')]$ has no solution for all actions c,c' such that $h \in \text{Cond}(c)$, $h' \in \text{Cond}(c')$, then it cannot be the case that the first instance of a usefully produces the required fluent h in P and the last instance of b usefully produces the required fluent h' in P.
(3) If h' is monotone*, then only the first instance of b can usefully produce h' in P. Hence there can only be one instance of action b in P, since we assume that the last instance of b usefully produces h in P. The result follows from the same argument as in case (2) with the extra constraint $\text{Once}(b)$ that there is only one instance of b in P.
The proof of part **(b)** of the lemma is similar.

Example 5. Consider the following EU temporal planning problem in which all actions are instantaneous:

a: p $\rightarrow$ f, e
b: p, e $\rightarrow$ g, $\neg$f
c: f $\rightarrow$ p

with I={f} and G={g}. One interpretation of these actions and fluents is: Have_Engine_checked (a), Drive (b),

Take_Petrol (c), Have_petrol (p), At_garage (f), Engine_OK (e), Arrived (g). The fluent f is not monotone since there is a plan c, a, b, a (in which the last action is clearly redundant) which establishes, destroys, and establishes f. However, f is –monotone* since in a minimal plan action a cannot usefully produce a fluent $h \in \text{Add}(a) = \{\text{f, e}\}$ after action b has destroyed f. In the case h=f, this is by Lemma 4(b)(2): c is the only action such that $\text{f} \in \text{Cond}(c)$, and TR[After($a$,$f$,$b$) $\cup$ For(a,last,f,c)] has no solution. (Note that since g, p are monotone by Lemma 1, we impose in TR that the actions b and c, which establish g and p, occur only once). In the case h=e, this is by Lemma 4(b)(3) since e is monotone* (by Lemma 1) and TR[After(a,f,b) $\cup$ Once(a)] has no solution.

The notions +monotone* and –monotone* allow us to define a tractable class of temporal planning problems which is considerably larger than the EU monotone class given by Cooper et al. [6]. We state without proof the following theorem, which follows immediately by the same argument as in the proof of the equivalent result for monotone (instead of monotone*) fluents [6], but this time considering only the minimal plans of a temporal planning problem.

Theorem 1. Given a positive temporal planning problem $\langle I,A,G\rangle$, define S' recursively to be the minimum set of fluents not in I which belong to G or to the conditions of some action which establishes a fluent of S'. Let $A^{\text{ind}} \subseteq A$ be the set of actions which establish at least one fluent in S'. Suppose that all constraints in Constr(A^{ind}) are interval constraints, the set of actions A^{ind} is establisher-unique relative to S', each fluent in $\text{Cond}(A^{\text{ind}}) \cup G$ is monotone* and each fluent in $I \cap (\text{Cond}(A^{\text{ind}}) \cup G)$ is –monotone*. Let TR* be a version of TR in which the +authorisation (–authorisation) constraints are applied to +monotone* (–monotone*) fluents. Then $\langle I,A,G\rangle$ has a temporal plan P if and only if

(1) $G \subseteq (I \backslash \text{Del}(A^{\text{ind}})) \cup \text{Add}(A^{\text{ind}})$
(2) $\text{Cond}(A^{\text{ind}}) \subseteq I \cup \text{Add}(A^{\text{ind}})$
(3) all $g \in G \cap \text{Del}(A^{\text{ind}}) \cap \text{Add}(A^{\text{ind}})$ are +monotone*
(4) TR* has a solution.

Theorem 2. Let Π^* be the class of positive temporal planning problems $\langle I,A,G\rangle$ in which all constraints in Constr(A) are interval constraints, A is EU, all fluents in $\text{Cond}(A) \cup G$ are monotone* and all fluents in $I \cap (\text{Cond}(A) \cup G)$ are –monotone*, where monotonicity* of all fluents can be deduced from Lemmas 1, 2, 3 and 4. Then Π^* is tractable.

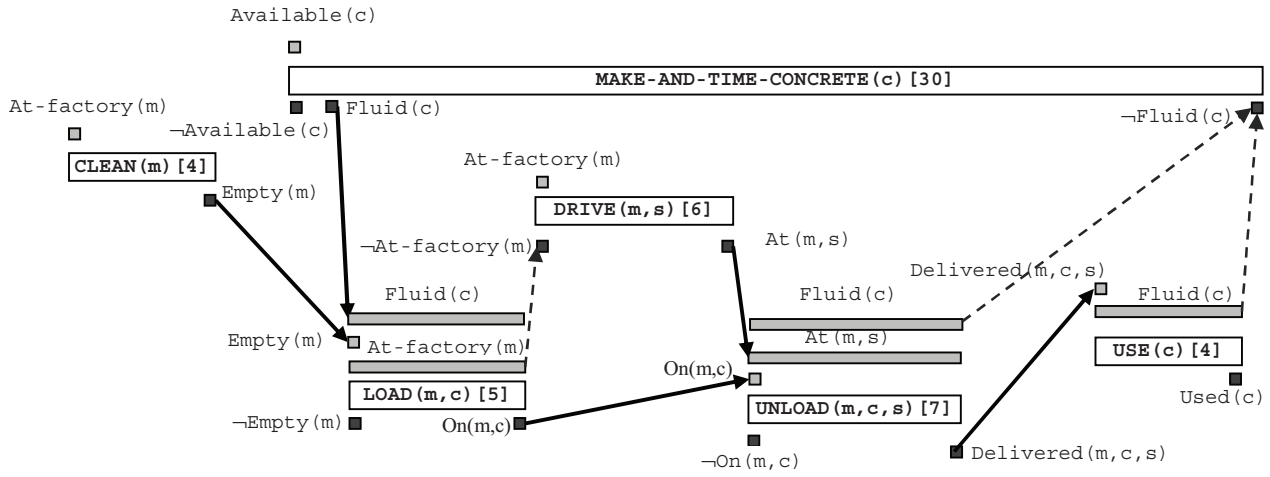
Proof: To prove tractability we have to give polynomial-time algorithms for both resolution and detection of temporal planning problems from the class Π^*.

We consider first the resolution of a temporal planning problem $\langle I,A,G\rangle$ in Π^*. Since A is establisher unique, in linear time we can find the set A^{ind} of actions described in the statement of Theorem 1. By establisher-uniqueness, all actions in A^{ind} are indispensable. We can check conditions (1)-(3) of Theorem 1 in linear time. We can then solve TR*, and hence $\langle I,A,G\rangle$, in $O(n^3)$ time and $O(n^2)$ space [12], where n is the total number of events in the actions in A.

The number of temporal relaxations to solve, in order to prove that a temporal planning problem belongs to Π^*, is proportional to the number of septuplets (a,f,b,c,h,c',h') such that $a,b,c,c' \in A$, $f \in \text{Add}(a) \cap \text{Del}(b)$, $h \in \text{Add}(a) \cap \text{Cond}(c)$ and $h' \in \text{Add}(b) \cap \text{Cond}(c')$. Assuming A is establisher-unique, the number of triples (a,f,b) satisfying $a,b \in A$ and $f \in \text{Add}(a) \cap \text{Del}(b)$ is bounded above by n. The number of pairs (c,h) such that $c \in A$ and $h \in \text{Cond}(c)$ is again bounded above by n. Therefore, the number of relaxations to be solved is $O(n^3)$. Each temporal relaxation can be solved in $O(n^3)$ time and $O(n^2)$ space [12]. It follows that the complexity of recognizing Π^* is $O(n^6)$ time and $O(n^2)$ space.

VII. Example of EU monotone* planning

Several examples of temporal planning problems from the chemical or pharmaceutical industries fall into the class of EU monotone problems [6]. However, as Examples 4 and 5 have shown, even in simple problems, fluents may be monotone* rather than monotone. We conclude this paper with the description of a planning domain in which we need to detect monotone* fluents in order to prove tractability.



The Temporal Cement Factory planning domain allows us to plan concrete mixing, delivery and use. An action of duration 30 time units *makes and times* a batch of concrete which is fluid from time unit 3 to 30 (after which it sets). At the same time, a concrete-mixer must be *cleaned*, in order for the concrete to be *loaded*, then *driven* to a building site, where it is *unloaded*. The concrete must then be *used* while it is still fluid. This set of actions A (illustrated in the plan shown in the figure) are all indispensable and the set of fluents appearing in the figure is the set S of sub-goals. The initial state I and the goal G are given by:

```
I = {At-factory(m), Available(c)},
G = {Delivered(m,c,s), Used(c)}
```

For all $a_i \neq a_j \in A$, we have Add(a_i) ∩ Add(a_j) ∩ $S = \emptyset$. Hence, by Definition 3, the set of actions A is EU relative to S. We can immediately remark that no actions delete the fluents Used(c), Delivered(m,c,s) and At(m,s), and no actions add the fluents Available(c) and At-factory(m). Thus, by Lemma 1, these fluents are both –monotone and +monotone. By Lemma 2, we can deduce that On(m,c) is –monotone since the temporal relaxation TR[After(LOAD(m,c),On(m,c),UNLOAD(m,c,s))] has no solution. Similarly, Fluid(c) is also –monotone by Lemma 2. We can deduce that Empty(m) is –monotone* by Lemma 4(b)(2), because the relaxation detects that a=CLEAN(m) cannot establish f=Empty(m), after b=LOAD(m, c) destroys Empty(m), and also usefully establish h=Empty(m) for c= LOAD(m, c). This is because in TR there is only one instance of LOAD(m, c) since Add(LOAD(m))={On(m,c)} and, as we have just seen, On(m,c) is monotone. We can now apply Theorem 1, since A is EU, all fluents are monotone* and all fluents in I are –monotone*. It follows that TR* is a solution procedure for this problem. The problem $<I,A,G>$ has a solution-plan, found by TR*, shown in the figure. We represent non-instantaneous actions by a rectangle. The duration of an action is given in square
brackets after the name of the action. Conditions are written above an action, and effects below. Causality constraints are represented by bold arrows, and –authorisation constraints by dotted arrows. This example can be extended to the case in which there are several sites, several batches of concrete and several mixers. It is monotone and remains EU provided that the goals (via the fluents Delivered(m,c,s)) specify which mixer m is to deliver which batch c to which building site s.

As another example involving monotone* fluents, consider the example given in a previous paper [6] involving the synthesis of a chemical using catalysers, where it was assumed that all catalysers are destroyed during the chemical reaction. However, if any of the catalysers are not destroyed by the chemical reaction they catalyze, then they can theoretically be used many times. Nevertheless, in a minimal plan, a second utilisation of a catalyser is not necessary and the rules given in Section V allow us to detect that all fluents are monotone*.

VIII. Conclusion

We have given a novel form of relaxation which can be used in temporal planning. It can detect unsolvable problems which cannot be detected by relaxations based on ignoring all destructions of fluents. It has applications in detecting indispensable actions and monotone fluents. This led to an extended notion of mononicity which allowed us to define a large tractable class of temporal planning problems whose recognition algorithm is based on our temporal relaxation.

References

[1] Bonet B., Loerincs G. and Geffner H. "A Robust and Fast Action Selection Mechanism for Planning", *AAAI-97/IAAI-97*, 714-719, 1997.

[2] Bylander T. "The Computational Complexity of Propositional STRIPS Planning". *Artificial Intelligence*, 69(1-2), 165-204, 1994.

[3] Coles A., Fox M., Long D. and Smith A. "Planning with Problems Requiring Temporal Coordination", *Proc. of AAAI 2008*, 892-897, 2008.

[4] Cooper M.C., de Roquemaurel M. and Régnier, P. "A weighted CSP approach to cost-optimal planning", *Artificial Intelligence Communications*, 24(1) 1-29, 2011.

[5] Cooper M.C., Maris F. and Régnier P. "Managing Temporal Cycles in Planning Problems Requiring Concurrency", *Computational Intelligence*, 29(1), 111-128, 2013.

[6] Cooper M.C., Maris F. and Régnier, P. "Tractable monotone temporal planning", *Proc. ICAPS 2012*.

[7] Cushing W., Kambhampati S., Mausam and Weld D.S. "When is Temporal Planning Really Temporal?", *IJCAI'2007*, 1852-1859, 2007.

[8] Do M.B. and Kambhampati S. "Sapa: A Multi-objective Metric Temporal Planner", *Journal of Artificial Intelligence Research* 20, 155-194, 2003.

[9] Eyerich P., Mattmüller R. and Röger G. "Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning", *Proc. ICAPS 2009*.

[10] Fox M. and Long D. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains", *Journal of Artificial Intelligence Research* 20, 61-124, 2003.

[11] Fox M., Long D. and Halsey K. "An Investigation into the Expressive Power of PDDL2.1", *Proc. of 16th European Conference on Artificial Intelligence*, pp. 328-342, 2004.

[12] Gerevini A. and Cristani M. "On Finding a Solution in Temporal Constraint Satisfaction Problems", *Proc. 15th International Joint Conference on Artificial Intelligence*, *IJCAI'1997*, 1460-1465, 1997.

[13] Ghallab M., Nau D.S. and Traverso P. *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.

[14] Jonsson P. and Bäckström C. "State-variable planning under structural restrictions: Algorithms and complexity", *Artificial Intelligence*, 100(1-2), 125-176, 1998.

[15] Koubarakis M. "Dense Time and Temporal Constraints with ≠", *KR'1992*, 24-35, 1992.

[16] Long D. and Fox M. "Exploiting a graphplan framework in temporal planning", *Proc. 13th International Conference on Automatic Planning and Scheduling*, 52-61, 2003.

[17] McDermott D. "PDDL, The Planning Domain Definition Language". *Technical Report*, http://cs-www.cs.yale.edu/ homes/dvm/, 1998.

[18] Rintanen J. "Complexity of concurrent temporal planning", *Proc. 17th ICAPS*, 280-287, 2007.

[19] Smith D.E. "The Case for Durative Actions: A Commentary on PDDL2.1", *Journal of Artificial Intelligence Research* 20, 149-154, 2003.