A Constraint-based Encoding for Domain-Independent Temporal Planning

Submission 88

–

Abstract. We present a general constraint-based encoding for domain-independent task planning. Task planning is characterized by causal relationships expressed as conditions and effects of optional actions. Possible actions are typically represented by templates, where each template can be instantiated into a number of primitive actions.
While most previous work for domain-independent task planning has focused on primitive actions in a state-oriented view, our encoding uses a fully lifted representation at the level of action templates. It follows a time-oriented view in the spirit of previous work in constraint-based scheduling.
As a result, the proposed encoding is simple and compact as it grows with the number of actions in a solution plan rather than the number of possible primitive actions. When solved with an SMT solver, we show that the proposed encoding is slightly more performent than state-of-the-art methods on temporally constrained planning benchmarks while clearly outperforming other fully constraint-based approaches.

1 Introduction

Task planning is a field of Artificial Intelligence concerned with finding a set of actions that would result in desirable state. The key difficulty in task planning lies in the handling of causal relationships between a large number of potential actions. Research in the field has focused primarily on the definition of relaxations of the classical planning problem in order to define accurate heuristic functions to guide tree search algorithms. Those heuristics have proved to be highly effective for reasoning on the causal relationships that occur in task planning and have driven most research to focus on state-based heuristic search.

Despite their success in classical planning, they have proved to be difficult to extend to more expressive models including time, resources or continuous changes. This lead to a renewal of interest into constraint-based models and search as a way to increase the expressiveness of domain-independent task planners [18,10,30].

In this paper, we propose an encoding for the causal relationships that are at the core of domain independent planning. Unlike the most recent work proposing compilations of domain-independent temporal planning to Constraint Satisfaction Problems that follows a state-oriented view [10,9], we use a time oriented

view inspired by the work in the field of constraint-based planning and scheduling [22,11,26,23]. The key benefit of this encoding is its simplicity and its good fit for integration into more general constraint satisfaction problems.

We start by giving a background on task planning and the representation of temporal planning problems as chronicles. We then show how such planning problems can be concisely encoded into Constraint Satisfaction Problems, using a fully lifted representation based on chronicles. The resulting encoding is leveraged in a fully domain-independent planner for the ANML language. Using an off-the-shelf SMT solver as a backend, the planner is shown to outperform state-of-the-art temporal planners on temporally constrained planning benchmarks. Last we discuss the related work in constraint-based planning, especially analyzing other domain-independent planners that rely on SMT.

2 Background

2.1 A Distilled Planning Problem

In its simplest formulation, a planning problem is composed of *(i)* an initial state, *(ii)* a goal state and *(iii)* a set of primitive actions.

A state is an assignment to a set of *state variables*, each denoting one particular feature of the the environment (e.g. the location of a truck). We denote as S the set of possible states.

A primitive action a is composed of:

- a set of conditions over state variables. It characterizes the set of states $S_a^{app} \subseteq S$ in which the action is applicable.
- a set of effects from which one can construct a state transition function $f_a : S_a^{app} \to S$

Given an initial state $s_0 \in S$, a goal state $s_g \in S$ and set of primitive actions, a plan is a sequence of primitive actions $\langle a_1, \ldots, a_n \rangle$. For given plan, one can build the resulting sequence of states $\langle s_1, \ldots, s_n \rangle$ such that $s_i = f_{a_i}(s_{i-1})$. A plan is a solution to a planning problem if all actions are applicable in the previous state ($\forall_{i \in [1,n]} s_{i-1} \in S_{a_i}^{app}$) and the final state is the goal state ($s_n = s_g$).

This formulation of planning as a state transition system has been at the core of domain independent planning research, and of PDDL, the de-facto standard language for describing planning problems [28]. Several extensions have been devised for extending the expressiveness of PDDL (most notably to handle a limited form temporal constructs and numeric variables [21,19]). Nevertheless, the focus of the language and the benchmarks of the International Planning Competition remains heavily focused on the encoding of the causal relationships that derive from the conditions and effects of primitive actions.

2.2 Temporal Planning as Chronicles

An other notable encoding for planning follows a time-oriented view as opposed to the state-oriented view above. We here detail the representation of planning

problems as chronicles [24], that was first introduced in the IxTeT planner [23]. This representation avoids direct references to states and instead describe the evolution of the environment through temporally qualified assertions over the state variables.

A *type* is defined by a set of values. A type is either *(i)* a set of domain constants (e.g. the type $Truck = \{ R_1, R_2 \}$ defines two truck objects R_1 and R_2 in the planning problem), *(ii)* a discrete or continuous set of numeric values, or *(iii)* a representation of temporal instants, for which we typically consider the set of natural numbers.[1]

A (decision) *variable* is associated with a type that defines its initial domain. We denote as *timepoints* the subset of variables that represent a temporal instant.

A *state variable* denotes the evolution of a particular state feature over time. A state variable is typically parameterized by one or multiple domain objects to represent the state of a particular object in planning problem. For instance $loc(R_1)$ denotes the evolution of the location of the truck R_1 over time. A state variable expression can be parameterized by decision variables, in which case the actual state variable it refers to depends on the value taken by its parameters, e.g., $loc(r)$ will refer to $loc(R_1)$ or $loc(R_2)$ depending on the value taken by the variable r of type $Truck$.

Chronicle In line with previous work in temporal planning, we represent the core constructs of a planning problem with chronicles [24]. A chronicle is a tuple $(V, X, C, E,)$ where:

- V is a set of variables.
- X is a set of constraint over the variables in V.
- C is a set of conditions. Each condition is of the form $[s, e]\ sv(p_1, \ldots, p_n) = v$ where s and e are timepoints in V, $sv(p_1, \ldots, p_n)$ is a parameterized state variable (with each $p_i \in V$) and $v \in V$ is a variable. A condition states that the state variable $sv(p_1, \ldots, p_n)$ must have the value v over the temporal interval $[s, e]$.
- E is a set of effects. Each effect is of the form $[s, e]\ sv(p_1, \ldots, p_n) \leftarrow v$ where s and e are timepoints, $sv(p_1, \ldots, p_n)$ is a parameterized state variable (with each $p_i \in V$) and $v \in V$ is a variable. Such an effect states that the state variable $sv(p_1, \ldots, p_n)$ will take the value v at time e. Over the temporal interval $]s, e[$ the state variable has an undefined value: it is transitioning from its previous value to v.

A chronicle is thus a constraint satisfaction problem (V, X) extended with additional constructs to represent the conditions and effects that are core to a planning problem.

[1] Note that this integer based representation of time is no less expressive than a real-valued representation when forbidding instantaneous changes, as common in temporal planning [15].

Action Chronicle A planning problem defines a set of action templates $\mathcal{T}$. Each action template $a \in \mathcal{T}$ can be instantiated into chronicles. We denote as $\mathcal{C}_a^i$ a chronicle instantiated from a, where i distinguishes chronicles instantiated from the same template.

Considering an action template $Go \in \mathcal{T}$ that moves a truck r from a location l_s to a location l_e, its instantiation as a chronicle $\mathcal{C}_{Go}$ could have the following components:

- $V = \{ r, l_s, l_e, t_s, t_e \}$ where r, l_s, l_e are variables representing the parameters of the action (truck, start location and end location) and t_s and t_e are timepoints representing the start and end times of the action.
- $X = \{ t_e = t_s + 10, l_s \neq l_e \}$, constraints stating that the action takes 10 time units and that the origin l_1 and destination l_2 must be different.
- $C = \{ [t_s, t_s]\ loc(r) = l_s) \}$, i.e., the truck r must be in location l_s at the action's start t_s.
- $E = \{[t_s, t_e]\ loc(r) \leftarrow l_e)\}$, i.e., the truck r will be in location l_e at the action's end t_e. Its location is undefined for the duration of the action $]t_s, t_e[$.

Such a chronicle is called an *action chronicle*.

Initial Chronicle We distinguish action chronicles from the initial chronicle $\mathcal{C}_0$ that represents the initial state and its expected evolution together with the planning objectives. Here specifying a problem where the truck R_1 is initially in location L_0 and must be in location L_2 or L_3 before time 100 is encoded with the chronicle $\mathcal{C}_0 = (V_0, X_0, C_0, E_0)$ where:

- $V_0 = \{ t, l \}$
- $X_0 = \{ t < 100,\ l = L_2 \vee l = L_3 \}$, constraints restricting the solution set.
- $C_0 = \{ [t, t]\ loc(R_1) = l \}$, condition specifying the goal.
- $E_0 = \{ [0, 0]\ loc(R_1) \leftarrow L_0 \}$, effect defining the initial state: the truck R_1 is in L_0.

A planning problem can be represented as a pair $(\mathcal{C}_0, \mathcal{T})$ where $\mathcal{C}_0$ defines the initial state and objectives and $\mathcal{T}$ is a set of actions templates that can be instantiated into action chronicles.

3 Planning as a Constraint Satisfaction Problem

We consider a *bounded* planning problem composed of an initial chronicle $\mathcal{C}_0$ and set of action chronicles AC. Each action chronicle $\mathcal{C}_a^i \in AC$ is associated to a boolean variable p_a^i that is true if $\mathcal{C}_a^i$ is part of the solution plan.

Consider the planning problem of moving the truck R to either $L2$ or $L3$ from the previous section. Considering a set of optional actions chronicle $\{ \mathcal{C}_{Go}^1, \mathcal{C}_{Go}^2 \}$, we can represent all the plans composed of 0, 1 or 2 instances of the Go action. The actual plan depends on the instantiation of the decision variables. One could build a satisfying solution from the partial assignment $\{p_{Go}^1 \leftarrow \top,\ r^1 \leftarrow R_1,\ l_s^1 \leftarrow L_0,\ l_e^1 \leftarrow L_2,\ p_{Go}^2 \leftarrow \bot \}$ where r^1, l_s^1 and l_e^2 being the eponymous variables in

$\mathcal{C}^1_{Go}$. Such an assignment states that only the first action would be present and would move the truck R_1 from L_0 to L_2. Note that this only partially defines the solution plan and other assignments would need to be made, notably to the action timepoints and to the variables of $\mathcal{C}_0$.

A *bounded* planning problem Π is the set of chronicles $\{\mathcal{C}_0\} \cup AC$ where each $\mathcal{C} \in \Pi$ is associated to a boolean variable $present(\mathcal{C})$ that is: p^i_a if $\mathcal{C}$ is an action chronicle $\mathcal{C}^i_a \in AC$; or $\top$ if $\mathcal{C} = \mathcal{C}_0$. This planning problem is called *bounded* as there is an upper limit on the number of actions in a solution plan defined by the number of action chronicles.

3.1 Building blocks

Given a bounded planning problem Π, we now describe the core structures for compiling it to a constraint satisfaction problem.

Condition Token Given a chronicle $\mathcal{C} \in \Pi$, each condition $[s, e]\ sv(p_1, \ldots, p_n) = v$ in $\mathcal{C}$ is associated to a *condition token*:

$$present(\mathcal{C}) : [s, e]\ sv(p_1, \ldots, p_n) = v$$

This token represent that, if $\mathcal{C}$ is part of the solution ($present(\mathcal{C}) = \top$), then the state variable $sv(p_1, \ldots, p_n)$ must have the value v over the temporal interval $[s, e]$.

We denote as C_Π the set of condition tokens in Π.

Effect Token Given a chronicle $\mathcal{C} \in \Pi$, we associate to each effect $[s, e]\ sv(p_1, \ldots, p_n) \leftarrow v$ in $\mathcal{C}$ an *effect token*:

$$present(\mathcal{C}) : [s, e, t]\ sv(p_1, \ldots, p_n) \leftarrow v$$

where t is an new timepoint variable. This token states that, if $\mathcal{C}$ is part of the solution ($present(\mathcal{C}) = \top$), then the state variable $sv(p_1, \ldots, p_n)$ is undefined over the temporal interval $]s, e[$, and has the value v over the temporal interval $[e, t]$. The introduction of the new decision variable t allows to encode a minimal time for which the effect will persist.

We denote as E_Π the set of effect tokens in Π.

Characteristics of Tokens Effect tokens here represent the evolution of state variables over time. Each (present) effect token impose a new value to its state variable. This value is constrained to be maintained on a given temporal interval. Effect token thus encode the state evolution. On the other hand, condition tokens place constraints on the state evolution by requesting a given state variable to have a given value over a temporal interval. Intuitively, such a condition is achieved if there is a corresponding effect token that imposes the appropriate value.

For a given token, the variables involved allow four degrees of freedom on which a solver might play to build a consistent plan:

- the presence of the token (variable $present(\cdot)$).
- the temporal interval on which it applies, (variables s, e and t).
- the state variable on which it applies (variables $p_1, \ldots, p_n$).
- the value taken by the state variable (variable v).

Those variables are typically shared with other tokens from the same action chronicle. These interdependencies between tokens are at the core of the hardness of planning as having an effect token to establish a condition requires the presence of all other tokens from the same chronicle. This in turn requires new conditions to be established as well as new effect tokens that might interact with the existing ones.

3.2 Constraints for Plan Consistency

Given a planning problem Π, we define a set of constraints that encode the requirements for a plan to be consistent.

Coherence constraint State variables are similar to unary resources in that they can only take a single value at a given time. Any two distinct effect tokens α and α' in E_Π must be coherent: they may not concurrently impose a value to the same state variable.

$$\begin{aligned} \text{Given } \alpha &= \langle p : [s, e, t]\ sv(p_1, \ldots, p_n) \leftarrow v \rangle \in E_\Pi \\ \alpha' &= \langle p' : [s', e', t']\ sv(p'_1, \ldots, p'_n) \leftarrow v' \rangle \in E_\Pi \end{aligned}$$

then the constraint $coherent(\alpha, \alpha')$ is defined as:

$$p \wedge p' \implies t \leq s' \vee t' \leq s \vee p_1 \neq p'_1 \vee \cdots \vee p_n \neq p'_n$$

The coherence constraint enforces that a given state variable has at most one value at any point in time. It is done by forcing two effect tokens to be non overlapping which can be enforced along three axis: presence, time and state variable.

Support constraint We say that a condition token $\beta \in C_\Pi$, is *supported by* an effect token $\alpha' \in E_\Pi$ if α' establishes the value required by β and that this value persists for the duration of β.

$$\begin{aligned} \text{Given } \beta &= \langle p : [s, e]\ sv(p_1, \ldots, p_n) = v \rangle \in C_\Pi \\ \alpha' &= \langle p' : [s', e', t']\ sv(p'_1, \ldots, p'_n) \leftarrow v' \rangle \in E_\Pi \end{aligned}$$

then $suported\text{-}by(\beta, \alpha')$ is defined as:

$$p' \wedge e' \leq s \wedge e \leq t' \wedge p_1 = p'_1 \wedge \cdots \wedge p_n = p'_n \wedge v = v'$$

Less formally, it means that β is supported by α' if *(i)* α' is present, *(ii)* β is contained in the interval $[e', t']$ over which the effect of α' persists, and *(iii)* α' establishes the right value on the right state variable.

The fact that a condition token $\beta \in C_\Pi$ must be supported by at least one effect token if it is present is encoded by the constraint *supported*(β):

$$\textit{present}(\beta) \implies \bigvee_{\alpha \in E_\Pi} \textit{supported-by}(\beta, \alpha)$$

Internal chronicle consistency Given an optional chronicle $\mathcal{C} = (V, X, C, E)$, if it is part of the solution then all its internal constraints must hold, which is represented by the constraint *consistent*($\mathcal{C}$):

$$\textit{present}(\mathcal{C}) \implies \bigwedge_{c \in X} c$$

Formulation as a CSP A bounded planning problem Π is encoded as a Constraint Satisfaction Problem (V_Π, X_Π) where:

$$\begin{aligned} V_\Pi &= \{\, V \mid (V, X, C, E) \in \Pi \,\} \\ &\quad \cup \{\, \textit{present}(\mathcal{C}) \mid \mathcal{C} \in \Pi \,\} \\ X_\Pi &= \{\, \textit{coherent}(\alpha, \alpha') \mid \alpha, \alpha' \in E_\Pi, \alpha \neq \alpha' \,\} \\ &\quad \cup \{\, \textit{supported}(\beta) \mid \beta \in C_\Pi \,\} \\ &\quad \cup \{\, \textit{consistent}(\mathcal{C}) \mid \mathcal{C} \in \Pi \,\} \end{aligned}$$

3.3 Symmetry breaking constraints

A given planning problem Π might contain several action chronicles for the same action template, e.g., several chronicles being instantiations of the *Go* action template. In the current formulation, nothing distinguishes two action chronicle of the same template and any satisfying assignment can be made into a new one by exchanging the variables of the two chronicles.

Given the set of action templates $\mathcal{T}$, for any action template $a \in \mathcal{T}$, we refer as $\langle \mathcal{C}_a^1, \ldots, \mathcal{C}_a^n \rangle$ all action chronicles of the template a in Π, in a arbitrary order.

From this ordering, we define two symmetry breaking constraints. The first one requires that, if a given chronicle is present, then all its predecessors in the ordering are present as well:

$$\bigwedge_{a \in \mathcal{T}} \bigwedge_{i \in [1, n-1]} \textit{present}(\mathcal{C}_a^{i+1}) \implies \textit{present}(\mathcal{C}_a^i)$$

The second one requires the ordering on chronicles to match the ordering of actions in the solution plan:

$$\bigwedge_{a \in \mathcal{T}} \bigwedge_{i \in [1, n-1]} \textit{start}(\mathcal{C}_a^{i+1}) \geq \textit{start}(\mathcal{C}_a^i)$$

where *start*($\mathcal{C}$) is the start timepoint of the action chronicle $\mathcal{C}$.

4 Instantiation in a Domain Independent Planner

We now describe how the presented encoding can be leveraged in a fully domain-independent temporal planner. We present LCP (Lifted Constraint-based Planner), a planner that solves ANML planning problem using an SMT solver as a backend. Rather than providing a fully fledged temporal planner, LCP aims at validating the encoding presented on standard temporal planning benchmarks.

4.1 The ANML Language

The Action Notation Modeling Language (ANML) is a recent proposal by NASA Ames Research Center to represent rich temporal planning and scheduling problems [31]. ANML features a clear notion of actions with conditions and effects integrated with a rich representation of time.

Compared to the temporal variants of PDDL, the ANML language has two important features. The support of multi-valued states variables, as opposed to the boolean state variables of PDDL, allows for a more compact representation of the state. In addition, ANML supports referencing timepoints others than the start and end of an actions, thus increasing the capabilities to model complex temporal actions. Temporal PDDL planning problems are usually easy to express in ANML, with some recent work to provide an automated translation [5,3].

Translation of ANML planning problems into chronicles is straightforward as all constructs we are interested in have a one to one mapping [6].[2] We parse an ANML problem file into the initial chronicle $\mathcal{C}_0$ and a set of action templates $\mathcal{T}$ that can be instantiated into action chronicles.

4.2 Solving with SMT

Our solving procedure works by incrementally generating bounded planning problems Π_k with an increasing number of optional actions. For each problem, an SMT solver is used to either prove the consistency of Π_k or the absence of satisfying assignment. If Π_k is consistent, the solution plan is extracted from the satisfying assignment. Otherwise the process continues with the next bounded problem Π_{k+1}.

Given an input planning problem $(\mathcal{C}_0, \mathcal{T})$, we generate a bounded problem Π_k using the procedure GenProblem$(\mathcal{C}_0, \mathcal{T}, k)$. as follows:

$$\Pi_k = \{ \mathcal{C}_0 \} \bigcup_{a \in \mathcal{T}} \{ \mathcal{C}_a^i \mid i \in [1, k] \}$$

For a problem Π_k, this formulation allows the presence of k instances of each action template $a \in \mathcal{T}$;

The overall procedure is given in Algorithm 1. The planner iteratively generates bounded planning problems of increasing depth k until a solution is found

[2] Omitted in our translations are the hierarchical and resource constructs of ANML that are beyond the scope of this paper.

or a max depth k_{max} is reached. The CHECKSMT(Π_k) encodes Π_k as a Constraint Satisfaction Problem using the encoding we presented. The consistency of the resulting the CSP is checked with the Z3 SMT solver [16].

Algorithm 1 Planning procedure of LCP (Lifted Constraint Planner) for an input problem $(\mathcal{C}_0, \mathcal{T})$ and maximum depth k_{max}.

```
function LCP(C_0, T, k_max)
    k ← 0
    while k ≤ k_max do
        Π_k ← GENPROBLEM(C_0, T, k)
        res ← CHECKSMT(Π_k)
        if res is a consistent model then
            return EXTRACTSOLUTION(res)
        end if
        k ← k + 1
    end while
    return failure
end function
```

5 Experiments

5.1 Comparison with state of the art temporal planners

We evaluate our approach against several state of the art temporal planners on benchmarks from the International Planning Competition (IPC). We focus on problems were time plays an important role in the solution: planning problems that have deadlines or time windows that constrain the occurrence time of actions in the plan. The rational for doing so is that planning problems without such features can be solved without any explicit handling of time (with the exception of some problem with required concurrency [14]).

We compare ourselves to Temporal Fast Downward (TFD) [20], OPTIC [2] and FAPE [18,6]. TFD is a forward-search state-space planner that ranked second in the last temporal track of the IPC. Its search is based on the addition of new primitives actions at the end of an existing plan and heavily relies on heuristics for guidance. OPTIC is similar in its use of forward search but uses a different heuristic and a lifted temporal representation handled in an STN. It is an improved version of POPF [13], runner up in the temporal track of the penultimate IPC.

FAPE represents another line of temporal planners that uses a constraint-based representation. FAPE plans in the space of lifted partial plans, where each plan is composed of a set of partially instantiated actions whose variables are embedded in a CSP. Search proceeds by either imposing values on variables or extending the partial plan with additional actions, thus extending the CSP with

new constraints, variables and values in previous domains. FAPE is the first planner in this line of work to show good performance in a domain-independent setting [6].

We use the *airport-timewindows*, *satellite-timewindows* and *pipesworld-deadlines* domains from IPC4 that feature durative actions and temporal constraints on the plan in the form of deadlines and time-windows in which actions can be performed. The *airport-timewindows* domain focuses on planning the takeoff and landing of several planes in an airport, the movement of planes being temporally constrained by the arrival of other planes. The *satellite-timewindows* domain plans observations and data transmission of a fleet of satellites subject to visibility windows. The *pipesworld-deadlines* domain focuses on planning the transportation of products through pipes, subject to delivery deadlines. TFD and OPTIC use the original PDDL domains while LCP and FAPE use their translation to ANML from FAPE's benchmark problems.[3]

As standard in the IPC, we evaluate the planners on the number of problems solved with a 30 minutes timeout. Results are given in Table 1. It can be seen that, on temporally constrained problems, LCP is slightly ahead of both FAPE and OPTIC in terms of number of problems solved. TFD performs poorly on such problems.

	LCP	FAPE	OPTIC	TFD
airport-timewindows	**8**	7	7	1
satellite-timewindows	4	**10**	4	0
pipesworld-deadlines	**17**	6	13	2
Total	**29**	23	24	3

Table 1. Number of problems solved by the considered planners within 30 minutes. Best result is given in boldface.

LCP performs best in the *pipesworld-deadlines* domains that is characterized by much interference between the available actions, reducing the ability to reason independently on goals. Its worse performance is on the *satellite-timewindows* domain whose difficulty lies in the number of mostly independent goals requiring a large number of actions. In this setting, LCP fails to prove the absence of solution before reaching the subproblem that actually contains one. The *airport-timewindows* domain is an intermediate between those two domains making LCP performs only slightly better than other planners.

Note on expressiveness Unlike PDDL temporal planners, LCP supports rich temporal constructs, including actions with conditions and effects besides the start and end timepoints of actions. It leverages ANML's multi-valued state

[3] `https://github.com/laas/fape/tree/master/planning/domains`

variables (as opposed to the boolean state variables of PDDL) to provide a more compact representation. Note that FAPE, that also uses the ANML language as input, has the same characteristics.

In addition, LCP readily supports numeric variables both as action parameters and state variables. State-of-the-art temporal planners do not support numeric parameters due to their need to ground the problem to derive heuristics, even though some recent work was done in this direction for forward-search planners [29].

Note on performance in non-temporal domains We also tested the performance of LCP on non-temporal domains, i.e., domains where time is either absent or limited to the duration of actions and can soundly be omitted [14]. On the domains tested (*blocks*, *logistics* and *rovers* from the IPC), LCP solved around two thirds of the problems solved by FAPE, OPTIC and TFD who all had similar performance. As could be expected, LCP does not reach the performance of state of the art planners on non-temporal problems. It nevertheless showed consistent performance in its handling of causal relationships.

5.2 Comparison with SMTPlan

Our work was motivated by the recent development in the planning community to extend the expressiveness of task planners. Research in domain independent planning has been mostly focused on state-based heuristic search. Such planners deliberately choose to cope with very large search spaces because this encoding allows the definition of very effective heuristic functions to guide its exploration.

The will to support more realistic problems through extensions to PDDL is however problematic. Indeed extensions affect the performance of heuristics that become less informative, with dramatic effects on search performance. This state of affairs has motivated the introduction of new planners that rely on SMT solvers to deal with the most expressive extensions of PDDL. Most notably, the most effective approaches for planning with PDDL+ rely on compilations to SMT [9,10].

SMTPlan+ [10] appears to be the most effective of those planners. It supports planning with continuous non-linear change and processes and the authors showed that those can be efficiently accounted for in an SMT representation.

Of all the planning benchmarks tested in the previous section, SMTPlan+ was only able to solve the two simplest instances of the *blocks* domain. Its performance on domain-independent temporal or non-temporal planning is thus largely below the ones of LCP and state-of-the-art planners.

SMTPlan+ uses a state-oriented representation where each state is associated to a given happening that alters the state (corresponding to the start or end of an action). Each state is represented by a set of boolean variables. To each happening is associated the choice of a primitive action that implies constraints on the happening's state variables.

To understand the implication of this representation, we compare the encoding of both LCP and SMTPlan+ on two problems of the *rovers* domains. The

two problems mainly differ by the number of objects in the domain, with a direct impact on the number of primitive actions. The first instance (p01) has only 63 primitive actions while the second instance considered (p10) has 382. Note that these numbers are low compared to standard planning benchmarks (e.g. the most difficult problem of the rovers domain has 32 437 reachable primitive actions).

The number of disjunctive constraints as well as the number of variables is given in Table 2 for both LCP and SMTPlan+. It can be seen that the size of the SMT formula at a given depth in SMTPlan+ is directly impacted by the number of primitive actions, making it quickly untractable even on problems of modest size. The size of the encoding grows linearly with the number of happenings (depth). On the other hand, LCP is mostly unaffected by the growth in problem size thanks to its use of partially instantiated actions. On the down side, the encoding of LCP grows quadratically with the depth. Note that, in the case of LCP, a depth of 4 means that each action template might be duplicated 4 times (in the *rovers* domain that has 9 actions templates, the plan at depth 4 might contain up to 36 actions). On the other hand, a durative action requires two happenings in SMTPlan+'s representation, i.e., a depth of 4 would only allow two sequenced actions.

	Rovers p01		Rovers p10	
Depth	LCP	SMTPlan+	LCP	SMTPlan+
1	109 / 115	1 987 / 332	165 / 132	28 760 / 1 828
2	295 / 218	3 987 / 664	399 / 235	59 293 / 3 656
3	561 / 321	6 161 / 996	713 / 338	89 196 / 5 484
4	907 / 424	8 248 / 1 328	1 107 / 441	119 099 / 7 312

Table 2. Number of disjunctive constraints (left) and variables (right) in the SMT formulas of LCP and SMTPlan+ at various depth. Depth is the number k of duplicated actions in LCP and the number of happenings in SMTPlan+.

6 Related Work

Graph-plan based encodings An historical application of constraint solvers to planning has been based on the graph-plan framework [8]. Such planners build a synthetic data structure that captures causal relationships between primitive actions up to a given plan length. While building such a data structure can be done in polynomial time, extracting a solution (or proving its absence) is combinatorial. CSP solvers have been leveraged to perform the plan extraction step in planners such as GP-CSP [17] and CSP-PLAN [27]. Both planners use a grounded state-based encoding where the value of variables in each state is

related to the previous state and primitive actions through a set of constraints. Despite recent work on improving the encoding of the CSP with table constraints [1], the performance of such planners has been superseded by forward search planners.

Plan-space and timeline-based planners Another line of research is the work on timeline-based planners [22,12,11] and lifted plan-space planners [23,6]. Both kind of planners use a time-oriented representation, and the chronicle model that we use originated in lifted plan-space planners [23]. Those planners search in the space of partial plans where the current partial plan is extended with new actions during search. Key aspects of the partial plan are represented in a CSP, including the actions' parameters and timepoints.

One difficulty is that the CSP is constructed online – both constraints and variables are added to the CSP – limiting the ability to use existing constraint solvers. More problematic is that the set of effects that can be used to support a given condition is not fixed *a priori* as new actions can be inserted in the partial plan. As a result, support constraints are often implemented in an *ad hoc* way, outside of the main constraint engine.

The internal representation of those planners is however closely related to the encoding proposed in this paper. Compared to plan-space planners, we avoid the explicit handling of threats by the introduction of a new timepoint in effect tokens to represent the minimal persistence of an effect. This additional timepoint is accounted for directly in the coherence constraint. This is important to limit the size of the CSP, as an *a priori* encoding of threats would be cubic in the number of possible actions.

The proximity with those planners might make it possible to adapt the relaxations, heuristics and propagators developed into global constraints in our setting, such as the one tailored to reason on causal relationships in EUROPA and FAPE [4,7].

SMT based planners The development of SMT encodings of planning problems in the last years has been motivated by the need to support richer planning problems, notably involving continuous change on numeric state variables in planners such as dReach [9] and SMTPlan+ [10]. As highlighted in the previous section, it is our feeling that the support of more complex problems has been done at the detriment of those planners scalability. Our objective with this paper is precisely to propose an alternative encoding that can *(i)* be used to efficiently represent planning problems and *(ii)* does not prevent its extension to problems with continuous change. Our encoding already supports continuous state variables and – unlike dReach and SMTPlan+ – action parameters with continuous domains. As for SMTPlan+ and dReach, our use of an SMT solver with non-linear arithmetic theories opens up the possibility of reasoning with non-linear continuous changes and processes.

Constraint-based Scheduling The consideration of optional activities in constraint solvers slowly bridges the gap that existed between planning and scheduling. Our formulation of a bounded planning problem is indeed very close to a scheduling problem with optional activities over given temporal intervals, in the spirit of those in CP Optimizer [26].

Achieving good, domain-independent, performance in a such a setting would certainly require specialized global constraints. This is a promising direction to tackle the current quadratic growth in the number of constraints of our representation.

Our coherence constraints are closely related to scheduling optional activities on unary resources for which existing techniques could be easily adapted [33,26,25]. The main difference here being the implicit choice of the resource (i.e. state variable) that is governed by several variables. Support constraints are more challenging and to our knowledge have no straightforward mapping to global constraints in scheduling. While plan-space planners do provide propagators for such support constraints [32,7], those are typically highly tailored to their search mechanism and adapting them would require more work.

7 Conclusion

In this paper, we presented a constraint-based encoding for temporal planning. The encoding is focused on handling the conditions and effects appearing in typical planning problems. It leverages a fully lifted representation and allows the addition of arbitrary constraints.

Its instantiation in a simple planner, using an off-the-shelf SMT solver, shows improved performance with respect to state of the art temporal planners on planning problems with deadlines and time windows. The resulting planner largely outperforms existing SMT based temporal planners on planning benchmarks.

References

1. Barták, R., Toropila, D.: Reformulating Constraint Models for Classical Planning. In: International Florida Artificial Intelligence Research Society Conference (FLAIRS) (2008)
2. Benton, J., Coles, A., Coles, A.: Temporal Planning with Preferences and Time-Dependent Continuous Costs. In: International Conference on Automated Planning and Scheduling (ICAPS) (2012)
3. Bernardini, S., Fagnani, F., Smith, D.E.: Extracting Lifted Mutual Exclusion Invariants from Temporal Planning Domains. Artificial Intelligence (2018)
4. Bernardini, S., Smith, D.E.: Developing Domain-Independent Search Control for EUROPA2. In: ICAPS Workshop on Heuristics and Search for Domain-independent Planning (HSDIP) (2007)
5. Bernardini, S., Smith, D.E.: Automatic Synthesis of Temporal Invariants. In: Symposium on Abstraction, Reformulation and Approximation (SARA) (2011)
6. Bit-Monnot, A.: Temporal and Hierarchical Models for Planning and Acting in Robotics. Ph.D. thesis, Université de Toulouse (2016)

7. Bit-Monnot, A., Smith, D.E., Do, M.B.: Delete-free Reachability Analysis for Temporal and Hierarchical Planning. In: European Conference on Artificial Intelligence (ECAI) (2016)
8. Blum, A.L., Furst, M.L.: Fast Planning through Planning Graph Analysis. Artificial Intelligence **90**(1–2) (1997)
9. Bryce, D., Gao, S., Musliner, D., Goldman, R.: SMT-based Nonlinear PDDL+ Planning. In: AAAI Conference on Artificial Intelligence (2015)
10. Cashmore, M., Fox, M., Long, D., Magazzeni, D.: A Compilation of the Full PDDL + Language into SMT. In: International Conference on Automated Planning and Scheduling (ICAPS) (2016)
11. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: Developing an End-to-End Planning Application from a Timeline Representation Framework. In: Innovative Applications of Artificial Intelligence Conference (IAAI) (2009)
12. Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., Tran, D.: ASPEN: Automated Planning and Scheduling for Space Mission Operations. In: International Conference on Space Operations (SpaceOps) (2000)
13. Coles, A., Coles, A., Fox, M., Long, D.: Forward-Chaining Partial-Order Planning. In: International Conference on Automated Planning and Scheduling (ICAPS) (2010)
14. Cushing, W., Kambhampati, S., Mausam, Weld, D.S.: When is Temporal Planning Really Temporal? In: International Joint Conference on Artificial Intelligence (IJCAI) (2007)
15. Cushing, W.A.: When is Temporal Planning Really Temporal? Ph.D. thesis, Arizona State University (2012)
16. De Moura, L., Bjørner, N.: Z3: An efficient SMT Solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2008)
17. Do, M.B., Kambhampati, S.: Solving planning-graph by compiling it into CSP. In: International Conference on Automated Planning and Scheduling (ICAPS) (2000)
18. Dvorák, F., Barták, R., Bit-Monnot, A., Ingrand, F., Ghallab, M.: Planning and Acting with Temporal and Hierarchical Decomposition Models. In: IEEE International Conference on Tools with Artificial Intelligence (ICTAI) (2014)
19. Edelkamp, S., Hoffmann, J.: PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. In: International Planning Competition (IPC-2004) (2004)
20. Eyerich, P., Mattmüller, R., Röger, G.: Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning. In: Springer Tracts in Advanced Robotics (STAR) (2012)
21. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research (JAIR) **20** (2003)
22. Frank, J., Jónsson, A.: Constraint-Based Attribute and Interval Planning. Constraints **8**(4) (2003)
23. Ghallab, M., Laruelle, H.: Representation and Control in IxTeT, a Temporal Planner. In: International Conference on Artificial Intelligence Planning and Scheduling (AIPS) (1994)
24. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning: Theory and Practice (2004)
25. Laborie, P., Rogerie, J., Shaw, P., Vilim, P.: Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources. In: International Florida Artificial Intelligence Research Society Conference (FLAIRS) (2009)

26. Laborie, P., Rogerie, J.: Reasoning with Conditional Time-Intervals. In: International Florida Artificial Intelligence Research Society Conference (FLAIRS) (2008)
27. Lopez, A., Bacchus, F.: Generalizing graphplan by formulating planning as a CSP. In: International Joint Conference on Artificial Intelligence (IJCAI) (2003)
28. McDermott, D., Ghallab, M., Howe, A.E., Knoblock, C.A., Ram, A., Veloso, M.M., Weld, D., Wilkins, D.E.: PDDL: the Planning Domain Definition Language. Tech. rep. (1998)
29. Savas, E., Fox, M., Long, D., Magazzeni, D.: Planning Using Actions with Control Parameters. In: European Conference on Artificial Intelligence (ECAI) (2016)
30. Scala, E., Ramirez, M., Haslum, P., Thiebaux, S.: Numeric Planning with Disjunctive Global Constraints via SMT. In: International Conference on Automated Planning and Scheduling (ICAPS) (2016)
31. Smith, D.E., Frank, J., Cushing, W.: The ANML Language. In: International Conference on Automated Planning and Scheduling (ICAPS) (2008)
32. Vidal, V., Geffner, H.: Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. Artificial Intelligence **170**(3) (2006)
33. Vilím, P., Barták, R., Čepek, O.: Extension of O(n log n) filtering algorithms for the unary resource constraint to optional activities. Constraints **10**(4) (2005)