Deciding Unsolvability in Temporal Planning under Action Non-Self-Overlapping

Stefan Panjkovic, Andrea Micheli, Alessandro Cimatti

Fondazione Bruno Kessler, Trento, Italy
{spanjkovic, amicheli, cimatti}@fbk.eu

Abstract

The field of Temporal Planning (TP) is receiving increasing interest for its many real-world applications. Most of the literature focuses on the TP problem of finding a plan, with algorithms that are not guaranteed to terminate when the problem admits no solution. In this paper, we present sound and complete decision procedures that address the dual problem of proving that no plan exists, which has important applications in oversubscription, model validation and optimization. We focus on the expressive and practically relevant semantics of action non-self-overlapping, recently proved to be PSPACE-complete. For this subclass, we propose two approaches: a reduction of the planning problem to model-checking of Timed Transition Systems, and a heuristic-search algorithm where temporal constraints are represented by Difference Bound Matrices. We implemented the approaches, and carried out an experimental evaluation against other state-of-the-art TP tools. On benchmarks that admit no plans, both approaches dramatically outperform the other planners, while the heuristic-search algorithm remains competitive on solvable benchmarks.

Introduction

AI Planning is the problem of synthesizing a course of actions that leads to a certain goal. It is a well-studied field of Artificial Intelligence, with a wide variety of real-world applications in logistics, autonomy, robotics, and industrial automation. *Temporal Planning* (TP) is concerned with temporal aspects, such as deadlines, synchronization between actions and the precise timing of events. Several temporal planners are able to find plans efficiently in a variety of domains. However, none of them is *guaranteed* to detect that the input problem is unsolvable, i.e. does not admit a solution plan. In fact, some temporal planning techniques (e.g. decision-epoch) are incomplete also in the case when a plan exists (Cushing et al. 2007), other temporal planning techniques based on heuristic search (e.g. (Coles et al. 2010; Simmons and Younes 2011)) explore a search space that is in general not finite and thus, in cases where the heuristic is unable to eventually prune all branches, the search algorithm diverges. Techniques based on reduction to satisfiability (e.g (Shin and Davis 2005; Rintanen 2007)) are oriented to find plans and are unable to detect that no plan exists.

Yet, being able to determine unsolvability has important applications. For example, when doing plan optimization, a method is to progressively tighten the problem constraints, trying to solve them in order to obtain more optimized plans. Clearly, when the constraints are tightened too much, the problem becomes unsolvable, and existing planners might diverge on this input. Another application is "oversubscription" planning (Smith 2004), where it might be impossible to satisfy all the goals. Detecting unsolvability is also important for domain validation (Khatib, Muscettola, and Havelund 2001).

The problem of detecting unsolvability of planning instances has been studied in the context of classical planning (i.e., where actions are instantaneous and no temporal constraints are present), and several techniques have been presented (e.g., (Suda 2014; Hoffmann, Kissmann, and Torralba 2014; Muise and Lipovetzky 2016)), but it only recently received theoretical attention in the realm of temporal planning. In this paper, we address unsolvability in TP from a practical point of view. In general, the complexity of TP depends on the domain of time. If time is interpreted as a discrete quantity, TP is EXPSPACE-complete (Rintanen 2007). Instead, if time is interpreted as a dense quantity, TP is undecidable (Gigante et al. 2020). In this paper, we concentrate on a very expressive and practically relevant subclass of TP, characterized by the assumption of action non-self-overlapping (ANSO), that is PSPACE-complete in both the dense and discrete cases (Gigante et al. 2020). We study sound and complete decision procedures for ANSO TP: if the problem is solvable then a plan is eventually found, and if the problem is unsolvable the algorithm terminates stating that no solution exists. We stress that current planning techniques are semi-decision procedures for TP, and thus they are incapable of guaranteeing termination even under the ANSO assumption.

We propose two novel terminating approaches: the first approach is a reduction to model-checking, which encodes the problem in a symbolic timed transition system (Cimatti et al. 2019) and uses the NUXMV (Cavada et al. 2014) model-checker; the second approach is a dedicated Forward-Chaining Temporal Planner called TAMER-CTP, which combines a heuristic-search schema inspired by (Valentini, Micheli, and Cimatti 2020) with a symbolic representation of time using Difference Bound Matrices (DBM).

We performed a thorough experimental evaluation on both solvable and unsolvable benchmarks, comparing our approaches with the state-of-the-art tools. In addition, we implemented an optimized UPPAAL encoding (Bengtsson et al. 1995) of the timed automaton designed by (Gigante et al. 2020) to prove PSPACE-completeness of ANSO TP. This was used in the experiments as a baseline for the unsolvable benchmarks. The results show that the encoding into NUXMV and TAMER-CTP are the best tools for unsolvable instances and exhibit complementary strengths; moreover, TAMER-CTP is also competitive with the state-of-the-art on solvable instances.

Temporal Planning under ANSO

We start by formalizing the TP problem; we tackle TP problems admitting Intermediate Conditions and Effects (ICE) (Valentini, Micheli, and Cimatti 2020) and we use the same fragment of the ANML (Smith, Frank, and Cushing 2008) modeling language used in (Valentini, Micheli, and Cimatti 2020) with two additional assumptions needed for the problem to be decidable. We forbid infinite-domain fluents and we work under the ANSO assumption: we disallow any plan that exhibits a time instant when two instances of the same ground action are running.

We interpret time over the rational numbers, and we define a timing as an expression that refers to the start or the end of an interval as follows.

Definition 1. *A timing is an expression of the form* START $+$ k *with* $k \in \mathbb{Q}_{\geq 0}$ *or* END.

Timings refer to the instant an action is started or ended; for example, a timing START$+4$ for an action starting at absolute time 3 and terminating at time 10, corresponds to time 7, while a timing END corresponds to time 10. Timings are also used to schedule Timed-Initial-Literals (TILs), where START corresponds to time 0 and END indicates the end of the plan.

We now define the abstract syntax of a planning problem.

Definition 2. *A planning problem P is a tuple $\langle F, T, A, G\rangle$:*
- *F is a finite set of Boolean and bounded-integer fluents;*
- *T is a finite set of TILs, each of the form $\langle [t]\ f := c\rangle$ where t is a timing, $f \in F$, and c is a constant;*
- *A is a set of actions of the form $\langle C, E, d\rangle$ where:*
  - *C is a set of conditions of the form $\langle [t_1, t_2], \phi\rangle$, where t_1 and t_2 are timings and ϕ is a Boolean expression on F;*
  - *E is a set of instantaneous effects of the form $\langle [t]\ f := c\rangle$ where $f \in F$, t is a timing and c is a constant;*
  - *d is an interval of possible durations, $d = [l, u]$ with $0 < l \leq u \leq \infty$;*
- *G is a set of timed goals of the form $\langle [t_1, t_2], \phi\rangle$, where t_1 and t_2 are timings and ϕ is a Boolean expression.*

A plan consists of a finite sequence of actions to be executed at specified times and each with a specified duration.

Definition 3. *A **time-triggered plan** π for P is a sequence $\langle\langle t_1, a_1, d_1\rangle, \langle t_2, a_2, d_2\rangle, ..., \langle t_n, a_n, d_n\rangle\rangle$, where $t_i \in \mathbb{Q}_{\geq 0}$ is the starting time, $a_i \in A$ is the action to be started, $d_i \in \mathbb{Q}_{\geq 0}$ is the action duration, and $t_i \leq t_{i+1}$.*

The formal semantics of the described language is presented in (Cimatti, Micheli, and Roveri 2017); for the sake of brevity we only report the main points. All the fluents of the problem have an assigned value at every time instant $t \geq 0$. For every fluent in F there must exist a TIL in T that assigns a value to the fluent at the instant $t = 0$, thus all fluents must be initialized to some value. A fluent always maintains its value until an action effect or a TIL modifies it. Two effects that modify the same fluent are said to be *mutually exclusive* (mutex). A plan that schedules two or more mutually exclusive events at the same time is considered invalid. An effect becomes visible *immediately after* the time it is scheduled to happen, while conditions are evaluated immediately.

A time-triggered plan π is a solution for a planning problem P if by simulating π on P, i.e. applying all the effects at the specified time relatively to the starting time of the action, and considering the induced trace that assigns a value to each fluent at each time, all the conditions are satisfied in their intervals, the duration of every action respects the constraints, and the plan execution yields a final state with no pending running actions and where the goal condition holds.

Every effect, condition start and condition termination of every action, as well as every TIL and every goal starting and ending is called an "event" of the planning problem. Intuitively, a plan execution can be seen as a sequence of events separated by *time elapses*. This is the same idea of "snap-actions" of PDDL 2.1 (Coles et al. 2008) generalized to the ANML case where we have events that might not coincide to the starting or ending of actions (called Intermediate Conditions and Effects (ICE) (Valentini, Micheli, and Cimatti 2020)).

Complexity and self-overlapping. The computational complexity of TP has been studied in (Rintanen 2007) and (Gigante et al. 2020). One of the most important aspects that influences the complexity is whether actions are permitted to self-overlap with already running instances of themselves.

Definition 4. *Given a planning problem $P = \langle F, T, A, G\rangle$ and a plan $\pi = \{\langle t_1, a_1, d_1\rangle, ..., \langle t_n, a_n, d_n\rangle\}$ for P, an action $a \in A$ is said to* self-overlap *in π if there exist any $1 \leq i, j \leq n$ such that $a = a_i = a_j$ and $t_i \leq t_j \leq t_i + d_i$.*

In this paper, we consider dense time and we assume that there is no self-overlap of actions. Gigante et al. (2020) proved that the problem is decidable and PSPACE-complete.

Model-Checking Decision Procedure for TP

The first approach that we present transforms a temporal planning problem P into an equivalent NUXMV model with clocks, which is a symbolic representation of a Timed Transition System (TTS) (Cimatti et al. 2019).

A TTS is defined by a set of discrete variables, a set of clock variables and a set of constraints which determine symbolically the transition relation of the model. Each constraint restricts the values that a variable can assume in the *next state*, given the value of variables in the *current state*. In a TTS, the transitions can be either discrete or time-elapses: in a discrete transition, all variables non-deterministically

modify their values in a way subject to the constraints of the transition relation (including clock variables); in a time-elapse, all clocks increase by the same amount, while discrete variables remain unchanged.

We have variables corresponding to the fluents of the problem (of type boolean or bounded integer according to the domain of the fluent), additional boolean variables to represent the fact that an action is running and that an intermediate event has been applied, and a clock for every action representing the time since the action was last started.

The transition relation of our encoding is specified by a constraint for the start and end of every action, stating that preconditions and duration constraints must hold, and for the application of every intermediate event, that can be applied only if the clock of the corresponding action is equal to their timing. Moreover, we encode a frame axiom (Shanahan 1997) to restrict variables from changing arbitrarily unless an effect is applied on them.

By verifying the invariant property corresponding to the negation of the goal condition, if a solution plan exists any counter-example trace shows how to reach a state where the goal condition holds. From it, by considering the time delays and the moments in which actions were started and ended, a plan can be reconstructed.

More formally, given a temporal planning problem $P \doteq \langle F, T, A, G \rangle$, the variables of the NUXMV encoding are the following:

- the fluent variables $\{f_1, f_2, \ldots, f_k\}$, one for each fluent in F with appropriate type to represent the domain of the fluent;
- the boolean *running* variables r_a, for every action $a \in A$;
- the clock variables c_a, for every $a \in A$;
- a global clock variable c_g;
- the boolean *count* variables $\{count_1, \ldots, count_p\}$, one for every event in P.

The *running* variable r_a indicates whether the action $a \in A$ is currently running, and if a is running then c_a represents the time since the action was last started. The global clock variable c_g represents the time since the start of the plan, and is used to schedule the execution of TILs and to check goals at the appropriate time. Finally, the *count* variable for an event is used to ensure the event is not skipped: the variable is set to *true* when the event is applied, and at the end of the action it is checked that the *count* variables of all the events of the action are *true*.

Fluent variables are initialized according to the initial values of the respective fluents, the *running* and *count* variables are initialized to *false*, and all the clocks are set to 0.

For every action $a \in A$, with interval of possible durations $d_a = [l_a, u_a]$, we impose that when the action is started (i.e. $\neg r_a \wedge next(r_a)$) its clock is reset to 0, and when it is ended its clock must satisfy the duration constraint and all the events of the action have been applied:

- $(\neg r_a \wedge next(r_a)) \rightarrow next(c_a) = 0 \wedge \bigwedge_{e \in a} \neg next(count_e)$
- $(r_a \wedge \neg next(r_a)) \rightarrow c_a \in [l_a, u_a] \wedge \bigwedge_{e \in a} next(count_e)$

For every event e that is an effect of the form $[\text{START} + k] f_i := v$ or a condition of the form $[\text{START} + k]\phi_i$ of an action a with $k > 0$, we specify that if a is running and its clock has the value of the timing of the event, then the event must be applied (we also set the corresponding *count* variable to *true*):

- $(r_a \wedge (c_a = k)) \rightarrow next(f_i) = v \wedge next(count_e)$
- $(r_a \wedge (c_a = k)) \rightarrow \phi_i \wedge next(count_e)$

Similarly, events attached to the END of an action a and events scheduled at START + 0, are encoded using $r_a \wedge \neg next(r_a)$ and $\neg r_a \wedge next(r_a)$ as left-hand side of the implication. Events e that are either TILs $[\text{START}+k] f_i := v$ or timed goals of the form $[\text{START}+k]\phi_i$ are handled similarly, using c_g instead of c_a:

- $(c_g = k) \rightarrow next(f_i) = v \wedge next(count_e)$
- $(c_g = k) \rightarrow \phi_i \wedge next(count_e)$

Finally, we need to specify the "frame axiom" (Shanahan 1997), because variables may change their value arbitrarily if we don't constrain them to change only when an effect is applied to them. For every clock variable c_a, corresponding to action a, we impose that the clock can change its value in a discrete transition only when the action is started: $(c_a \neq next(c_a)) \rightarrow (\neg r_a \wedge next(r_a))$. As the global clock c_g is never reset, we add the following constraint to avoid any changes during discrete transitions: $c_g = next(c_g)$. For every fluent variable f_i we consider the set of effects/TILs modifying the fluent $\{[t_1] f_i := v_1, \ldots, [t_m] f_i := v_m\}$, and we impose an axiom: $(f_i \neq next(f_i)) \rightarrow e_1 \vee \ldots \vee e_m$, where e_i is an expression checking that the i-th effect/TIL is being applied in this transition.

A goal state is a state in which the goal condition holds, no action is running and all the TILs and goals have been applied. To determine whether a goal state can be reached, we check the following invariant property:

INVARSPEC $\neg(\bigwedge_g \phi_g \wedge \bigwedge_{a \in A} \neg r_a \wedge \bigwedge_i gcount_i)$,

where g are the end goals $[\text{END}]\phi_g$ and $\{gcount_1, \ldots, gcount_s\}$ is the set of *count* variables relative to TILs and goals with timing START + k. The above property is essentially the negation of a goal state. If the property is violated, i.e. there exists a path that reaches a goal state, then the returned counterexample trace can be converted to a valid solution plan for the original temporal planning problem P.

Theorem 1. *The encoding of a temporal planning problem P admits a counter-example if and only if P is solvable.*

Proof. (Sketch) The TTS encoding captures the semantics of ANSO temporal planning as derived from (Gigante et al. 2020). Any trace of the TTS corresponds to a valid trace in P. We exploit ANSO semantics by creating exactly one clock and one running variable for each action (without ANSO, we would need an unbounded number of those clocks and variables). The invariant property in the TTS directly states that the problem is unsolvable, hence there exists a counterexample if and only if the planning problem is solvable. □

Note that TTSs as per (Cimatti et al. 2019) are in general undecidable, but we target a fragment that does not use infinite-state variables except for clocks and thus the fragment targeted by our encoding is analogous to a timed automaton and the reachability model-checking problem is decidable.

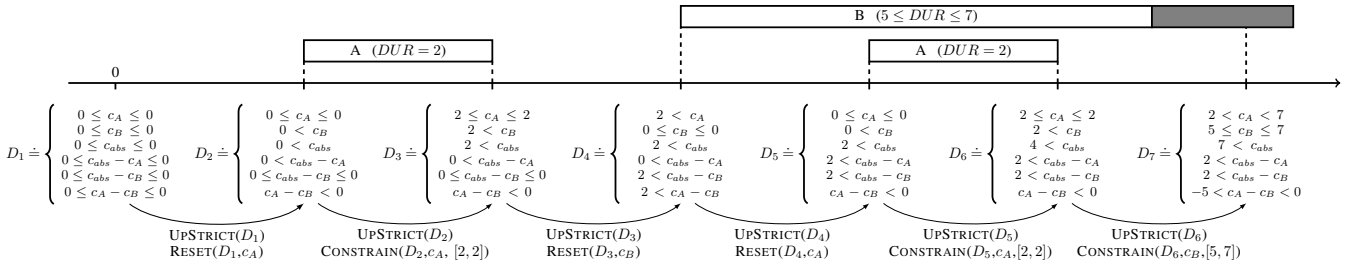

Figure 1: Example of DBM usage in TAMER-CTP. The temporal constraints in the example plan depicted as a Gantt chart are captured by the sequence of DBM constraints on the clocks c_{abs} (the absolute clock, i.e. the time since the plan has started), c_a and c_b.

Heuristic Search Decision Procedure for TP

We now present the TAMER-CTP planning algorithm, a complete heuristic-search based temporal planner that works by maintaining the temporal information symbolic with Difference Bound Matrices (DBMs) (Bengtsson and Yi 2004), while using explicit propositional states. TAMER-CTP inherits the subdivision of actions in time-points, the search procedure and the heuristic from TAMER-FTP (the planner presented by Valentini, Micheli, and Cimatti 2020), but it differs completely in the way it handles temporal constraints. While TAMER-FTP relies on Simple Temporal Networks (STNs) (Dechter, Meiri, and Pearl 1991) to add and check constraints between time-points, TAMER-CTP is based on DBMs, the data structure used in the context of TA to represent zones (Bengtsson et al. 1995). STNs can be used to schedule plans by creating a time point for each event in the plan and then adding appropriate temporal constraints among these time points. Differently, DBMs represent a region of possible timings for a fixed number of *clocks*. In order to construct a plan, we can introduce one clock for each action, that measures the time elapsed since the action was last started; then, we associate a DBM to each search state that maintains the temporal constraints relevant for the construction of successor states.

Consider the example plan in Figure 1. With an STN, we can use time points to represent its temporal constraints (the reference time-point z and two time points for each action instance x: x_s representing the action start and x_e representing the termination): $z < A^1_s < A^1_e < B_s < A^2_s < A^2_e < B_e \wedge A^1_e - A^1_s = 2 \wedge A^2_e - A^2_s = 2 \wedge 5 \leq B_e - B_s \leq 7$. Using DBMs we introduce one clock for each action (c_A and c_B) and one for the absolute time that is never reset c_{abs}. Differently from the STN, we cannot represent all the temporal constraints of the plan using a single DBM, but we represent the temporal information at each state with a dedicated DBM: each DBM can be derived from the previous one in the sequence. In Figure 1, we start with the DBM where all clocks are 0 and then we let time elapse. When we start action A, we *reset* the clock c_A to 0 obtaining the second DBM. When an action is terminated, we enforce the duration constraints in the DBM (e.g. we force $c_A = 2$ in the third DBM). If at any point, the DBM becomes empty (i.e. unsatisfiable), then the plan is temporally unfeasible.

The key feature of DBMs is that they support subsumption checking, i.e. checking whether the time region represented by a DBM is a subset of the one represented by another DBM. This is a crucial operation in order to obtain a terminating algorithm. This operation is pivotal to check if all the plan suffixes from a certain state are also valid suffixes from another state, allowing us to prune the first state as it is dominated by the other. On the contrary, checking the subsumption of states using STNs is problematic: as detailed in (Coles and Coles 2016) if two states are reached with two permutations of the same set of actions, one can solve a NP-hard graph isomorphism problem to check if the first STN subsumes the other. However, there is no work we are aware of that tackles the problem of checking subsumption of STN temporal states that are achieved by different sets of actions. In fact, STNs do have one time-point for each step in the plan prefix leading to the state, and so two STNs with different prefixes are not directly comparable.

The way TAMER-CTP works is inspired by explicit-state model-checkers for TA: we use DBMs to ensure that the sequence of events explored by the planner are schedulable, meaning that it is possible to assign a timing to each event that respects all the temporal constraints. We also use the "normalization" procedure (Bengtsson and Yi 2004) to finitize the set of possible DBMs for a problem. TAMER-CTP saves every state encountered during the search to check if all the possible plans starting from a new state u are already covered by the ones starting from a visited state v. This is done by checking that the assignment to the state variables is the same in u and v, and by ensuring that the DBM of u is subsumed by the DBM of v. This allows TAMER-CTP to recognize that a state does not need to be further expanded as any plan starting from it will be already taken into account, hence ensuring termination.

Difference Bound Matrices. A DBM is a finite and canonical representation of a conjunction of atomic clock constraints over a set of clocks. Each constraint is of the form $x - y \preceq k$, where x and y are clocks, $\preceq \in \{<, \leq\}$ and $k \in \mathbb{Q}$ (Bengtsson and Yi 2004). Given a DBM D, the following transformations and checks are used by our algorithm.

- **IsConsistent**(D) checks that the solution set of D is non-empty. During state-space exploration this is used to remove inconsistent states that are not schedulable.
- **Up**(D) computes the strongest postcondition of a zone with respect to delay, i.e. the zone representing all clock

assignments that can be reached from D by waiting. A similar operation, **UPSTRICT**(D), also makes lower bounds strict, meaning that all clocks are delayed by a strictly positive amount of time.
- **RESET**(D, c) assigns the value 0 to the clock c.
- **CONSTRAIN**$(D, c, [t_1, t_2])$ adds the $t_1 \leq c \leq t_2$ constraint to the DBM D.
- **SUBSUMED**(D_1, D_2) checks whether D_2 is subsumed in D_1, i.e. any assignment satisfying D_2 also satisfies D_1.
- **NORMALIZE**$(D, \vec{k})$ applies the $k-$normalization procedure (Bengtsson and Yi 2004) to finitize the set of possible DBMs for a given problem.

Planning Algorithm. Hereinafter, we assume a planning instance $P \doteq \langle F, T, A, G \rangle$ is given and we define the search schema employed by TAMER-CTP. The search state structure is the same as the one used by TAMER-FTP, except for σ that in TAMER-CTP is a DBM instead of an STN.

Definition 5. *A **search state** is a tuple $\langle \mu, \delta, \lambda, \sigma \rangle$ where:*
- *μ is the assignment to all the fluents of the problem;*
- *δ is the set of active durative conditions that need to be maintained valid in this state;*
- *λ is a list of lists of events, one for every action plus one for TILs and goals;*
- *σ is a DBM that symbolically maintains the temporal constraints. It has one clock for every action, representing the time since the action was last started plus the absolute clock c_{abs}, representing the time since the plan started.*

Each list in λ contains the events that still need to be executed for a particular action: it constitutes an "agenda" of future commitments to be resolved. Each time a new action is started, its list of events is added to a new list in λ and each time an event is consumed, it is removed from λ. Since we are working under the ANSO assumption, the size of λ in each state is bounded: $|\lambda| \leq N_a + 1$, where N_a is the number of ground actions in the problem. Hence, we cannot have multiple instances of the same action concurrently running.

Definition 6. *The **initial state** $\langle \mu_0, \emptyset, \lambda_0, \sigma_0 \rangle$ is as follows:*
- *μ_0 is the initial assignment for every fluent;*
- *λ_0 has only one list that is filled with the ordered "agenda" of TILs and timed goals to be resolved;*
- *σ_0 is the zero DBM, where every clock is set to 0.*

Next we define the set of successors of a given state that are obtained by either applying an event or starting a new action in the state. For each state generated in this way, we can optionally perform a *time elapse* step to advance the time. The successor function is outlined in Algorithm 1. Given a state s, GETSUCCESSORS iterates through every applicable action and event in the current state and creates a new state n. The applicability functions check action preconditions and conditions of events on $s.\mu$, enforce mutual exclusion constraints, and ensure that the timings are compatible with $s.\sigma$ (e.g. if we executed an event of action a at START + 5, it is impossible to select an event at time START + 7 of the same action without a time-elapse in between). Moreover, APPLICABLEEVENTS ensures that two events belonging to the same list in λ and having the same

Algorithm 1: The pseudo-code of TAMER-CTP

```
1  procedure SEARCH( )
2      V ← NEWSET( )
3      Q ← NEWPRIORITYQUEUE( )
4      PUSH(Q, GETINITSTATE(), HADD(GETINITSTATE()))
5      while s ← POPMIN(Q) do
6          if CHECKNOSUBSUMPTION(V, s) then
7              if ISGOAL(s) then
8                  return GETPLAN(s)
9              ADDTOSET(V, s)
10             for all ss ∈ GETSUCCESSORS(s) do
11                 if CHECKNOSUBSUMPTION(V, ss) then
12                     PUSH(Q, ss, G(ss) + HADD(ss))
13     return No Plan Exists
14 procedure GETSUCCESSORS(s)
15     succ ← NEWSET( )
16     for all a ∈ APPLICABLEACTIONS(s) do
17         n ← COPYSTATE(s)
18         n.λ ← n.λ + GETORDEREDEVENTS(a)
19         RESET(n.σ, a)
20         ADDTOSET(succ, n)
21     for all ev ∈ APPLICABLEEVENTS(s) do
22         n ← APPLYEVENT(s, ev, GETCLOCK(ev))
23         if ISCONSISTENT(n.σ) then
24             ADDTOSET(succ, n)
25     for all n ∈ succ do
26         n′ ← COPYSTATE(n)
27         UPSTRICT(n′.σ)
28         ADDTOSET(succ, n′)
29     return succ
30 procedure APPLYEVENT(s, ev, clk)
31     ⟨μ, δ, λ, σ⟩ ← COPYSTATE(s)
32     if ev is [START + k, ···]φ then          ▷ Start of durative condition
33         CONSTRAIN(σ, clk, [k, ∞])
34         ADDTOSET(δ, ev)
35     else if ev is [··· , START + k]φ then    ▷ End of durative condition
36         CONSTRAIN(σ, clk, [−∞, k])
37         REMOVEFROMSET(δ, ev)
38     else if ev is [START + k]f := v then                    ▷ Effect
39         CONSTRAIN(σ, clk, [k, k])
40         μ[f] ← v
41     else if ev is [END, END]φ then                ▷ End condition
42         CONSTRAIN(σ, GETCLOCK(ev), DURCONSTRAINT(ev))
43     else if ev is [··· , END]φ then          ▷ End of durative condition
44         CONSTRAIN(σ, GETCLOCK(ev), DURCONSTRAINT(ev))
45         REMOVEFROMSET(δ, ev)
46     else if ev is [END]f := v then                    ▷ End Effect
47         CONSTRAIN(σ, GETCLOCK(ev), DURCONSTRAINT(ev))
48         μ[f] ← v
49     λ ← REMOVEEVENT(λ, ev)
50     NORMALIZE(σ, GETMAXCLOCKVALUES())
51     return ⟨μ, δ, λ, σ⟩
```

timings are not separated by a time-elapse: this can be done by looking at the path leading to s. The algorithm imposes the action duration constraints (i.e. $l_a \leq c_a \leq u_a$ where $[l_a, u_a]$ are the duration constraints of action a) at lines 34, 36 and 39, where events lined to the END are considered. Crucially, we forbid two instances of the same action to run in parallel by making a not applicable if an event of a is still in $s.\lambda$, ensuring the ANSO semantics. For each generated state n, we also construct the state n' where a time elapse is applied to n. This is done by calling the UPSTRICT() operation on the DBM $n'.\sigma$. Note that it is sufficient to consider just one time elapse, because for any DBM D, UPSTRICT(UPSTRICT(D)) = UPSTRICT(D) (i.e., letting time pass twice is the same as letting time pass once, because the amount of elapsed time is symbolic). APPLYEVENT creates a successor state by consuming an event in λ. If the

event is a condition or an effect the clock of the corresponding action is constrained with the timing of the event. Effects are applied to μ and durative conditions are added or removed from δ. Normalization is also applied using the maximal upper bounds for each clock; those are computed by taking the maximum upper bound of any event in the corresponding action; the bound for c_{abs} is the maximum timing among all goals and TILs.

Given this search schema, we employ an exploration algorithm based on A* (Hart, Nilsson, and Raphael 1968). Every state s has two values: $\mathrm{G}(s)$ that is the length of the path from the initial state to s, and $\textsc{Hadd}(s)$ that is the heuristic value of s computed using the standard h_{add} heuristic (Bonet and Geffner 2001) applied to the classical planning relaxation of the problem presented in (Valentini, Micheli, and Cimatti 2020). As usual in A*, we expand the state s in the queue Q with the lowest value of $(\mathrm{G}(s) + \textsc{Hadd}(s))$. When a state s is chosen, we first check whether there is a memoized state that has the same propositional part (μ, λ and δ) and whose DBM subsumes the DBM of the considered state: if that is the case then s is discarded as its successors are a subset of the successors of s, otherwise we add the considered state to the set of visited states V. If the state is a goal, meaning that the agenda λ is empty, then the exploration concludes and a plan is returned, otherwise its successors are computed, and all those that are not subsumed by an already visited state are added to Q. If the queue of states becomes empty, then all the states that are reachable from the initial state have been explored, and the procedure can declare that no solution exists. A plan can be reconstructed from the sequence of states leading from the initial state to a goal state by constructing an STN using the same approach in (Valentini, Micheli, and Cimatti 2020).

Theorem 2. TAMER-CTP *is a sound and complete decision procedure for ANSO TP.*

Proof. (Sketch) The states reachable by the TAMER-CTP exploration are finite because of the DBM k-normalization procedure and are in a bisimulation relation with the search states of a reachability TA solver for the encoding defined in (Gigante et al. 2020) (duly extended to support ICE). If a plan exists, then TAMER-CTP eventually finds it, because the TA solver would eventually find it. Vice-versa, if no plan exists, then TAMER-CTP cannot return a plan because there is no reachable state that is a goal for the TA and, due to the bisimulation, no goal state for TAMER-CTP exists. Full proofs are available in (Panjkovic, Micheli, and Cimatti 2022). □

Experimental Evaluation

We implemented the NUXMV translation and the TAMER-CTP algorithm in C++ using a custom library to handle DBMs and we experimentally evaluated their merits. In order to have a baseline to evaluate the performance of our approaches on unsolvable benchmarks, we implemented an optimized UPPAAL encoding (Bengtsson et al. 1995) of the timed automaton reduction that was used in the theoretical work of (Gigante et al. 2020) to prove PSPACE-completeness of ANSO TP. The encoding was also extended to support ICE, as is the case for NUXMV and TAMER-CTP. (full encoding details, all implementations and benchmarks are available in (Panjkovic, Micheli, and Cimatti 2022)).

We considered a comprehensive set of benchmarks and various state-of-the-art planners, namely OPTIC (Benton, Coles, and Coles 2012) (using the the "container-action construction" (Smith 2003) to support ICE), TPACK (Micheli and Scala 2019), ANMLSMT, a SATPlan-like SMT encoding for ANML, and TAMER-FTP (Valentini, Micheli, and Cimatti 2020): all these planners eventually find a plan when one exists, but are not guaranteed to terminate if the problem is unsolvable.

We consider both solvable and unsolvable problems. We sourced the solvable benchmarks from (Valentini, Micheli, and Cimatti 2020): the set includes standard temporal IPC instances (Vallati et al. 2015) (without ICE), IPC instances augmented with temporal uncertainty and reduced to temporal problems with ICE (Cimatti et al. 2018), and two "industrial" domains (MAJSP and PAINTER), both modeled using ICE. For the unsolvable case we crafted four domains. We modified the MAJSP domain to have an infeasible battery requirement and the PAINTER domain to require an impossible synchronization of operations. For MAJSP, we considered two different, but equivalent, formulations of the same problem (indicated as MAJSP 1 and MAJSP 2). We also created a new domain, called SYNC, that requires two actions to start and terminate into specific time windows that are too narrow. All these domains are unsolvable because either the resources are too constrained or the required timing and synchronization are impossible. Moreover, we considered the MATCHCELLAR IPC domain (Vallati et al. 2015) and made it unsolvable by reducing the number of available "matches" in each instance.

All the experiments were performed on a Core i9-9900KS with 1800s/20GB of time and memory limit. To better interpret and explain the results we also consider virtual solvers, that are fictitious planners obtained by combining the performance of other planners. The "Virtual best Solver" (VBS(S)) for a set of solvers S takes the minimum time needed to solve each benchmark across the solvers, simulating a parallel execution. All the benchmarks and the implementation of the presented approaches are available in (Panjkovic, Micheli, and Cimatti 2022).

Results. Figure 2 summarizes the experimental results. The coverage table reports the results for each unsolvable domain and the aggregated solvable results. The table and the cactus plot illustrate that the NUXMV approach performs really well in the unsolvable case, solving the highest number of instances followed by TAMER-CTP. The state-of-the-art planners cannot solve unsolvable instances except for a few: those are instances where the (incomplete) heuristic pruning is able to eventually detect unsolvability. It is evident that those approaches are not designed to deal with unsolvable benchmarks as they diverge even on very small instances. The cactus plot indicates that combining NUXMV and TAMER-CTP essentially gives the VBS(All) performance, showing that these two approaches are sufficient to cover virtually all the benchmarks. This is supported by the

Domain	OPTIC	TPACK	ANMLSMT	TAMER-FTP	UPPAAL	NUXMV	TAMER-CTP (hadd)	TAMER-CTP (blind)	VBS(TAMER-CTP,NUXMV)	VBS(All)
Total Solvable	**96**	**244**	**201**	**398**	**26**	**28**	**390**	**38**	**390**	**569**
MAJSP 1 (uns.)	1	1	0	0	0	22	11	12	26	28
MAJSP 2 (uns.)	0	0	0	0	47	117	87	82	133	133
MATCHCELLAR (uns.)	3	0	0	2	3	2	2	2	2	3
PAINTER (uns.)	2	0	0	0	4	3	5	4	5	5
SYNC (uns.)	0	0	0	0	4	7	5	5	7	7
Total Unsolvable	**6**	**1**	**0**	**2**	**58**	**151**	**110**	**105**	**173**	**176**
Total Overall	**102**	**245**	**201**	**400**	**84**	**179**	**500**	**143**	**563**	**745**

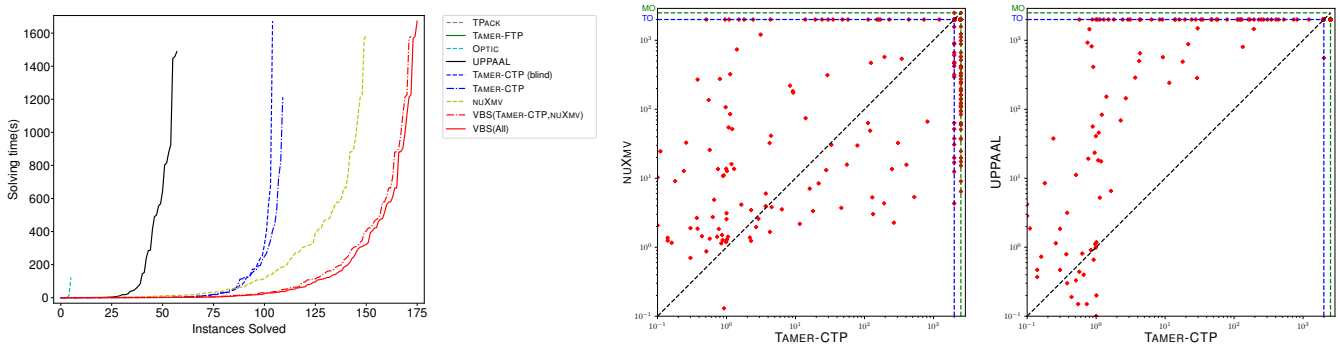

Figure 2: (Above) Coverage table showing aggregate results for solvable benchmarks and detailed for unsolvable benchmarks. (Below) Plots for the unsolavable benchmarks: cactus plot (left) and scatter plots comparing TAMER-CTP with NUXMV (center) and UPPAAL (right).

scatter plots highlighting the complementarity and by the fact that UPPAAL is dominated by TAMER-CTP: TAMER-CTP can solve all the problems that are solved by UPPAAL (and more), while NUXMV and TAMER-CTP work well on different sets of problems. This is interesting because it also highlights a complementarity between the fully symbolic approach employed by NUXMV, and the explicit-state heuristic search that is used by TAMER-CTP. We believe this complementarity is analogous to the one it is observed in the verification community between explicit-state and symbolic model-checkers. We analyzed the instances where TAMER-CTP and NUXMV excel respectively, but we could not find a simple discriminating factor predicting which technique would be the winner. A deeper analysis on this is left for future work.

The first row of the table aggregates the coverage for the solvable benchmarks. In this case, the native heuristic-search approach employed by TAMER-CTP outperforms by far the compilation-based approaches that rely on NUXMV and UPPAAL. This is expected, as the guidance provided by the heuristic on a solvable problem during the search can speed up substantially the resolution of the problem, avoiding the exploration of unpromising areas of the state space. Moreover, the results show that TAMER-CTP remains competitive with state-of-the-art planners on solvable instances, even though its primary purpose was to guarantee termination on unsolvable instances.

Finally, we studied the impact of the h_{add} heuristic: the TAMER-CTP (blind) column in Figure 2 indicates a modification of TAMER-CTP where the heuristic always returns 0. The heuristic is fundamental in the solvable case to guide the search, but the pruning capabilities are beneficial also in the unsolvable case: TAMER-CTP (blind) is faster in state expansion, but h_{add} reduces the search space size of TAMER-CTP, resulting in an overall better performance even in the unsolvable case.

Conclusion

We considered the class of temporal planning problems disallowing action self-overlapping and we presented two novel sound-and-complete decision procedures for it: a reduction to symbolic model-checking of timed systems and the first implementation of a heuristic-search temporal planner that is *guaranteed* to terminate on unsolvable instances. To the best of our knowledge, these are the first practical temporal planners that guarantee termination on any instance, following the theoretical work by (Gigante et al. 2020) that proved PSPACE-completeness for this class of problems. Our experiments show that these two approaches are far superior to other planners in proving unsolvability, and that the proposed heuristic-search algorithm scales well also on solvable instances, compared to the state-of-the-art.

In the future, we will use our approaches to develop optimizing TP procedures and we will investigate the synergies with (Rintanen 2017) to increase the performance of our algorithm by reducing the number of DBM clocks.

Acknowledgments

The authors are grateful for the partial support to this research by AIPlan4EU, a project funded by the European Commission - Horizon 2020 research and innovation programme under grant agreement number 101016442.

References

Bengtsson, J.; Larsen, K. G.; Larsson, F.; Pettersson, P.; and Yi, W. 1995. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In Alur, R.; Henzinger, T. A.; and Sontag, E. D., eds., *Hybrid Systems III: Verification and Control*, volume 1066 of *LNCS*, 232–243. Springer.

Bengtsson, J.; and Yi, W. 2004. Timed Automata: Semantics, Algorithms and Tools. In *Lectures on Concurrency and Petri Nets. LNCS*, volume 3098.

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*.

Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; and Tonetta, S. 2014. The nuXmv Symbolic Model Checker. In Biere, A.; and Bloem, R., eds., *CAV*, volume 8559 of *LNCS*, 334–342. Springer.

Cimatti, A.; Do, M.; Micheli, A.; Roveri, M.; and Smith, D. E. 2018. Strong temporal planning with uncontrollable durations. *Artif. Intell.*, 256: 1–34.

Cimatti, A.; Griggio, A.; Magnago, E.; Roveri, M.; and Tonetta, S. 2019. Extending nuXmv with Timed Transition Systems and Timed Temporal Properties. In *CAV*, 376–386.

Cimatti, A.; Micheli, A.; and Roveri, M. 2017. Validating Domains and Plans for Temporal Planning via Encoding into Infinite-State Linear Temporal Logic. In *AAAI*.

Coles, A.; and Coles, A. 2016. Have I Been Here Before? State Memoization in Temporal Planning. In *ICAPS*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *ICAPS*, 42–49.

Coles, A.; Fox, M.; Long, D.; and Coles, A. 2008. Planning with Problems Requiring Temporal Coordination. In *AAAI*.

Cushing, W.; Kambhampati, S.; Mausam, M.; and Weld, D. 2007. When is Temporal Planning Really Temporal? In *IJCAI*, 1852–1859.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence*, 49: 61–95.

Gigante, N.; Micheli, A.; Montanari, A.; and Scala, E. 2020. Decidability and Complexity of Action-Based Temporal Planning over Dense Time. In *AAAI*.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 441–446. IOS Press.

Khatib, L.; Muscettola, N.; and Havelund, K. 2001. Mapping Temporal Planning Constraints into Timed Automata. In *TIME*, 21–27. IEEE Computer Society.

Micheli, A.; and Scala, E. 2019. Temporal Planning with Temporal Metric Trajectory Constraints. In *AAAI*.

Muise, C.; and Lipovetzky, N. 2016. 2016 Unsolvability International Planning Competition. https://unsolve-ipc.eng.unimelb.edu.au.

Panjkovic, S.; Micheli, A.; and Cimatti, A. 2022. Deciding Unsolvability in Temporal Planning under Action Non-Self-Overlapping: Additional Material. https://es.fbk.eu/people/amicheli/resources/aaai22.

Rintanen, J. 2007. Complexity of Concurrent Temporal Planning. In *ICAPS*, 280–287.

Rintanen, J. 2017. Temporal Planning with Clock-Based SMT Encodings. In Sierra, C., ed., *IJCAI*, 743–749. ijcai.org.

Shanahan, M. 1997. *Solving the frame problem - a mathematical investigation of the common sense law of inertia*. MIT Press. ISBN 978-0-262-19384-9.

Shin, J.-A.; and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence*, 166: 194–253.

Simmons, R.; and Younes, H. 2011. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20.

Smith, D.; Frank, J.; and Cushing, W. 2008. The ANML language. In *KEPS*.

Smith, D. E. 2003. The case for durative actions: A commentary on PDDL2. 1. *Journal of Artificial Intelligence Research*.

Smith, D. E. 2004. Choosing Objectives in Over-Subscription Planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *ICAPS*, 393–401. AAAI.

Suda, M. 2014. Property Directed Reachability for Automated Planning. *J. Artif. Intell. Res.*, 50: 265–319.

Valentini, A.; Micheli, A.; and Cimatti, A. 2020. Temporal Planning with Intermediate Conditions and Effects. In *AAAI*.

Vallati, M.; Chrpa, L.; Grzes, M.; Mccluskey, T.; Roberts, M.; and Sanner, S. 2015. The 2014 International Planning Competition: Progress and Trends. *Ai Magazine*, 36: 90–98.