Temporal Planning with Intermediate Conditions and Effects

Alessandro Valentini, Andrea Micheli, Alessandro Cimatti
Fondazione Bruno Kessler, Trento, Italy
{alvalentini,amicheli,cimatti}@fbk.eu

Abstract

Automated temporal planning is the technology of choice when controlling systems that can execute more actions in parallel and when temporal constraints, such as deadlines, are needed in the model. One limitation of several action-based planning systems is that actions are modeled as intervals having conditions and effects only at the extremes and as invariants, but no conditions nor effects can be specified at arbitrary points or sub-intervals.

In this paper, we address this limitation by providing an effective heuristic-search technique for temporal planning, allowing the definition of actions with conditions and effects at any arbitrary time within the action duration. We experimentally demonstrate that our approach is far better than standard encodings in PDDL 2.1 and is competitive with other approaches that can (directly or indirectly) represent intermediate action conditions or effects.

1 Introduction

Automated temporal planning concerns the synthesis of strategies to reach a desired goal with a system that is formally specified by providing an initial condition together with the possible actions that can drive it in presence of temporal constraints. In this context, actions become intervals (instead of being instantaneous as in classical planning) that have a duration (possibly subject to metric constraints). Similarly, plans are no longer simple sequences of actions, but they are schedules. Automated temporal planning received considerable attention in the literature, and the definition of the standard PDDL 2.1 language (Fox and Long 2003) fueled the research of effective search-based techniques to solve the problem (Coles et al. 2010; Eyerich, Mattmüller, and Röger 2012; Rankooh and Ghassem-Sani 2015).

One limitation of many approaches encountered in several practical applications is that conditions and effects of each action can be only defined when the action starts or terminates, or as an invariant condition over all the action duration. This has been recognized (Smith 2003; Cimatti et al. 2018) as one of the major limitations of the PDDL 2.1 language, even if some compilation-based approaches are known. For example, this is crucial for the modeling of deadlines that are introduced as a consequence of an action.

Relaxing this limitation means allowing Intermediate Conditions and Effects (ICE): hence permitting the definition of actions with conditions being checked at arbitrary (possibly punctual) sub-intervals within the action duration and with effects happening at arbitrary points within the action interval. Notably, the ANML (Smith, Frank, and Cushing 2008) planning formalism offers this feature, but few planners support this. Our goal in this paper is to natively offer effective support for planning in domains with ICE.

The contributions of this paper are twofold. First, we present a heuristic-search planning technique able to solve temporal planning problems with ICE. Planners based on heuristic search techniques are currently the state of the art in several areas of planning; however, none is supporting ICE. We fill this gap by defining a suitable search space, generalizing the lifted-time approach of POPF (Coles et al. 2010) and providing a powerful relaxation of the problem for automatic heuristic computation. Second, we present an automated code-generation technique that produces a quasi-domain-dependent planner from a given model of the system. In particular, we define a method that maintains the generality of a model-based, domain independent technique, while providing some of the computational advantages of domain-dependent implementations. We read a planning instance with ICE and generate an executable embedding our search technique, specialized for the characteristics of the instance. The resulting executable is time-efficient and can solve a range of different problems without recompilation.

We experimentally evaluate the proposed technique by implementing it in a planner called TAMER and comparing against state of the art tools. The comparison comprises domains from the literature and domains inspired by industrial projects where time and temporal constraints are key aspects and the use of ICE facilitates the modeling. Our results show that our technique, thanks to the native support for ICE, is significantly faster and is able to solve many more problems than the state of the art tools on the industrially-inspired domains. We also experimentally evaluated the quasi-domain-dependent planner generation: our analysis shows that the approach scales even better than TAMER.

2 Problem Definition

Syntactically, we define a planning problem with ICE analogously to usual action-based temporal planning problems –

e.g. (Fox and Long 2003) – but we allow for conditions and effects to be specified at times relative to the start or the end of the action instance they belong to.

Definition 2.1. *An **ICE effect** on predicate p at relative time τ is a tuple $\langle \tau, p \rangle$ where τ is either* START $+ k$ *or* END $- k$ *with $k \in \mathbb{Q}_{>=0}$. An **ICE condition**[1] on predicate p in the relative interval $[\tau_1, \tau_2]$ is a tuple $\langle [\tau_1, \tau_2], p \rangle$ where τ_i is either* START $+ k_i$ *or* END $- k_i$ *with $k_i \in \mathbb{Q}_{>=0}$.*

Intuitively, an ICE effect (that can be an add or a delete) is applied at the time specified by τ that is relative to the start or the ending time of the action this effect belongs to. Similarly, conditions are checked on the closed intervals with extremes relative to the interval extremes of the action they belong to. In addition, we support timed-initial-literals (TILs) (Gerevini et al. 2009) expressed as a set of ICE effects where START refers to the beginning of time (i.e. to time 0) and END indicates the end of the plan execution (i.e the makespan). Similarly, we allow for timed goals (both instantaneous and durative) as a set of ICE conditions with the same interpretation of START and END.

Definition 2.2. *A **problem with ICE** is a tuple $\langle P, A, I, T, G \rangle$ where P is a finite set of Boolean predicates; A is a set of actions, each action a has a minimal (d_a^{min}) and maximal d_a^{max} duration, a set of ICE conditions C_a, a set of add ICE effects E_a^+ and a set of delete ICE effects E_a^- (with $E_a^+ \cap E_a^- = \emptyset$); $I \subseteq P$ is the initial state; T is a set of ICE effects partitioned in add (T^+) and delete (T^-), representing TILs; and G is a set of ICE goal conditions.*

A plan for an ICE planning problem is analogous to a plan for a standard temporal planning problem: we have a finite set of action instances to be executed, each having a specified starting time t and a specified duration d.

Definition 2.3. *A **plan** π is a finite set of tuples $\langle t, a, d \rangle$ where $t \in \mathbb{Q}_{>=0}$; $a \in A$; and $d \in \mathbb{Q}_{>0}$ with $d_a^{min} \leq d \leq d_a^{max}$. The makespan is $ms_\pi \doteq max(\{t+d \mid \langle t, a, d \rangle \in \pi\})$.*

Semantically, a plan is valid if the execution of the plan respects all the action conditions and goals and if no two contradicting effects are applied concurrently.

Definition 2.4. *The set of add (resp. delete) effects of a plan π for a planning problem with ICE $\mathcal{P} \doteq \langle P, A, I, T, G \rangle$ is the set $E^+_{\pi,\mathcal{P}}$ (resp. $E^-_{\pi,\mathcal{P}}$) defined as the union of the following sets:*
- $\{\langle t+k, p \rangle \mid \langle t, a, d \rangle \in \pi, \langle$ START $+k, p \rangle \in E_a^+\}$ *(resp $\in E_a^-$);*
- $\{\langle t+d-k, p \rangle \mid \langle t, a, d \rangle \in \pi, \langle$ END $-k, p \rangle \in E_a^+\}$ *(resp $\in E_a^-$);*
- $\{\langle k, p \rangle \mid \langle$ START $+ k, p \rangle \in T^+\}$ *(resp $\in T^-$);*
- $\{\langle ms_\pi - k, p \rangle \mid \langle$ END $- k, p \rangle \in T^+\}$ *(resp $\in T^-$).*

We define $E_{\pi,\mathcal{P}}$ as $E^+_{\pi,\mathcal{P}} \cup E^-_{\pi,\mathcal{P}}$.

Intuitively, we assign a time to all the effects of the plan actions and the TILs. Similarly, we can collect all the conditions imposed by either the plan actions or the timed goals.

Definition 2.5. *The set of conditions of a plan π for a planning problem with ICE $\mathcal{P} \doteq \langle P, A, I, T, G \rangle$ is the set $C_{\pi,\mathcal{P}}$ defined as the union of the following sets:*
- $\{\langle [t+k_1, t+k_2], p \rangle \mid \langle t, a, d \rangle \in \pi, \langle [$START$+k_1,$ START$+k_2], p \rangle \in C_a\}$;
- $\{\langle [t+k_1, t+d-k_2], p \rangle \mid \langle t, a, d \rangle \in \pi, \langle [$START$+k_1,$ END$-k_2], p \rangle \in C_a\}$;
- $\{\langle [t+d-k_1, t+k_2], p \rangle \mid \langle t, a, d \rangle \in \pi, \langle [END-k_1,$ START$+k_2], p \rangle \in C_a\}$;
- $\{\langle [t+d-k_1, t+d-k_2], p \rangle \mid \langle t, a, d \rangle \in \pi, \langle [END-k_1,$ END$-k_2], p \rangle \in C_a\}$;
- $\{\langle [k_1, k_2], p \rangle \mid \langle [$START$+k_1,$ START$+k_2], p \rangle \in G\}$;
- $\{\langle [k_1, ms_\pi - k_2], p \rangle \mid \langle [$START$+k_1,$ END$-k_2], p \rangle \in G\}$;
- $\{\langle [ms_\pi - k_1, k_2], p \rangle \mid \langle [END-k_1,$ START$+k_2], p \rangle \in G\}$;
- $\{\langle [ms_\pi - k_1, ms_\pi - k_2], p \rangle \mid \langle [END-k_1,$ END$-k_2], p \rangle \in G\}$.

We can now define the semantics of the ICE problem by first explaining what is a trace induced by a plan and then imposing the validity conditions on such a trace.

Definition 2.6. *A **trace** of $\mathcal{P} \doteq \langle P, A, I, T, G \rangle$ for a plan π and a predicate $p \in P$ is a function $V_p : \mathbb{Q}_{>=0} \to \mathbb{B}$, assigning a Boolean value to p at each time:*
- *$V_p(0) \doteq \top$ if $p \in I$; $V_p(0) \doteq \bot$ if $p \notin I$;*
- *$V_p(t) \doteq V_p(0)$ if $t \leq min(\{w \mid \langle w, p \rangle \in E_{\pi,\mathcal{P}}\})$;*
- *$V_p(t) \doteq \top$ if $\langle max(\{w \mid w < t, \langle w, p \rangle \in E_{\pi,\mathcal{P}}\}), p \rangle \in E^+_{\pi,\mathcal{P}}$;*
- *$V_p(t) \doteq \bot$ if $\langle max(\{w \mid w < t, \langle w, p \rangle \in E_{\pi,\mathcal{P}}\}), p \rangle \in E^-_{\pi,\mathcal{P}}$.*

Definition 2.7. *A **plan** π **is valid** for $\mathcal{P} \doteq \langle P, A, I, T, G \rangle$ if $E^+_{\pi,\mathcal{P}} \cap E^-_{\pi,\mathcal{P}} = \emptyset$ and for each condition $\langle [t_1, t_2], p \rangle \in C_{\pi,\mathcal{P}}$, $V_p(k) = \top$ for all $t_1 \leq k \leq t_2$.*

3 Heuristic Search for ICE

We present our heuristic-search method for solving planning problems with ICE by first presenting the search-space design and then a relaxation used to compute heuristic values. In the following, we assume a temporal planning problem with ICE $\mathcal{P} \doteq \langle P, A, I, T, G \rangle$ is given.

The general idea behind the engineering of our search-space is to maintain the temporal information symbolic while using explicit "propositional" states, similarly to planners such as POPF (Coles et al. 2010). Ideally, we separately consider effects and condition startings/endings as atomic "events" that change the state of the search. We encode such events as *time-points*.

Definition 3.1. *A **time-point** is either: $\langle \pi^\vdash \rangle$, indicating the start of plan instant; $\langle \pi^\dashv \rangle$, indicating the end of plan instant; $\langle act^\vdash, a, id \rangle$, with $a \in A$ and $id \in \mathbb{N}$, indicating the time at which an instance of a identified with id is started; $\langle act^\dashv, a, id \rangle$, with $a \in A$ and $id \in \mathbb{N}$, indicating the time at which an instance of a identified with id is terminated; $\langle cnd^\vdash, c \rangle$, with $c \in P$, indicating the time at which a durative condition starts; $\langle cnd^\dashv, c \rangle$, with $c \in P$, indicating the time at which a durative condition ends; $\langle cnd^[], c \rangle$, with $c \in P$, indicating the time at which an instantaneous condition is checked; $\langle eff^+, p \rangle$, with $p \in P$, indicating the time at which an add effect takes place; $\langle eff^-, p \rangle$, with $p \in P$, indicating the time at which a delete effect takes place. We write $kind(t)$ for the first element of a time-point t.*

Definition 3.2. *The **timed time-points of a set of add (resp. delete) effects** E^+ are $ttp(E^+) \doteq \{\langle \tau, \langle eff^+, p \rangle \rangle \mid \langle \tau, p \rangle \in E^+\}$ (resp. $ttp(E^-) \doteq \{\langle \tau, \langle eff^+, p \rangle \rangle \mid \langle \tau, p \rangle \in E^+\}$).*
*The **timed time-points of a set of conditions** C are $ttp(C) \doteq \{\langle \tau, \langle cnd^[], p \rangle \rangle \mid \langle [\tau, \tau], p \rangle \in C\} \cup \{\langle \tau_1, \langle cnd^\vdash, p \rangle \rangle, \langle \tau_2, \langle cnd^\dashv, p \rangle \rangle \mid \langle [\tau_1, \tau_2], p \rangle \in C, \tau_1 \neq \tau_2\}$.*

Given an action $a \in A$, we define the set of timed time-points of a as $ttp(a) \doteq ttp(C_a) \cup ttp(E_a^+) \cup ttp(E_a^-)$. Sim-

[1] We only formalize closed condition intervals; open and semi-open intervals are supported by our implementation.

ilarly, we define the set of goals and TILs timed time-points as $ttp(T, G) \doteq ttp(T^+) \cup ttp(T^-) \cup ttp(G)$.

Without loss of generality, from here on, we assume that for each action a in the ICE planning problem and for each duration, the relative ordering of time-points in $ttp(a)$ is fixed for every $d \in \mathbb{Q}_{>0}$ such that $d_a^{min} \leq d \leq d_a^{max}$. It is easy to see that if this is not the case, we can split the interval $[d_a^{min}, d_a^{max}]$ in sub-intervals having such a property and create a copy of action a with appropriate duration constraints for each sub-interval. The number of actions created by this transformation is at most quadratic, because having n timings as START $+ k$ and m specified as END $- k$, we can construct all the possible total orderings respecting two total orderings of size n and m, that are at most $m \times n$.

The search proceeds by either starting new action instances, thus adding new time-points (corresponding to the just-started action events) in a "todo-list", or by consuming such time-points by applying their effects and checking their conditions on the state. In this way, we construct a total order of time-points that is causally-valid. In order to symbolically maintain and check the temporal constraints we use a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991).

Definition 3.3. *A **search state** is a tuple $\langle \mu, \delta, \lambda, \chi, \omega \rangle$ s.t.:*
- *$\mu \subseteq P$ records the predicates that are true in the state;*
- *δ is a multiset of predicates in P, representing the active durative conditions to be maintained valid;*
- *λ is a list of lists of time-points. It constitutes the "agenda" of future commitments to be resolved. λ contains a list for TILs and goals having a timing relative to the plan start, one for those that are timed relative to the plan end, and a list for each action that has been started. Each list contains the time-points that have not been explored by the search yet. Crucially, all the lists are sorted according to the total order of time-points.*
- *χ is an STN defined over time-points that stores and checks the metric and precedence temporal constraints;*
- *ω is the last time-point evaluated in this search branch.*

The initial state for our problem is defined according to algorithm 1. Intuitively, we start from the propositional initial state I, enriched with an STN in which the time-points corresponding to all the TILs in T and the goals in G are prepared and constrained to their prescribed timings. In order to enforce such constraints, we create two special time-points, $\langle \pi^{\vdash} \rangle$ and $\langle \pi^{\dashv} \rangle$ representing the plan execution beginning and ending moments, respectively. All the constraints for TILs and goals are expressed relatively to these time-points by directly translating the ICE effects and conditions expressions as STN constraints. All the time-points being created are collected in two sets sl and el that contain the time-points whose timing is relative to the start of the plan and to the end of the plan, respectively. These sets are used to initialize the λ agenda with two ordered lists that contain the time-points to be expanded. The reason why we use two lists is because we have a direct total order between the time-points that are relative to START, and another total ordering for the time-points constrained with END, but we do not know how these time-points are interleaved. By using two lists we basically force the search to non-deterministically select one of the top elements in λ to expand.

Algorithm 1 Initial state computation

```
1: procedure FillTN(t⊢, t⊣, E+, E−, C, χ)
2:     sl, el ← ∅, ∅
3:     for all ⟨τ, t⟩ ∈ ttp(C) ∪ ttp(E+) ∪ ttp(E−) do
4:         if τ = START + k then
5:             sl ← sl ∪ {⟨k, t⟩}
6:             PushTNConstraint(χ, t − t⊢ = k)
7:         else if τ = END − k then
8:             el ← el ∪ {⟨k, t⟩}
9:             PushTNConstraint(χ, t⊣ − t = k)
10:    return ⟨sl, el⟩
11: procedure GetInit( )
12:    χ ← MakeEmptyTN( )
13:    sl, el ← FillTN(⟨π⊢⟩, ⟨π⊣⟩, T+, T−, G, χ)
14:    λ ← [ SortByAscendingTime(sl), SortByDescendingTime(el) ]
15:    return ⟨I, ∅, λ, χ, ⟨π⊢⟩⟩
```

Given a search state $s = \langle \mu, \delta, \lambda, \chi, \omega \rangle$ (according to definition 3.3), we define the set of successors of s (indicated as SUCC(s)) as the following set of states:

$$\{\textsc{SuccTp}(s, tp) \mid \langle tp, \cdots \rangle \in \lambda\} \cup \{\textsc{SuccAct}(s, a) \mid a \in A\}$$

We have two kind of successors, namely the ones deriving from the evaluation of time-points in λ, by selecting one list and expanding its head, and the ones obtained by starting new actions instances. The expansion of the selected time-point tp by means of the SUCCTP function in algorithm 2 is obtained by either applying the effects (if tp represents an effect, lines 4-7) or by checking the conditions (if tp represents conditions, lines 8-11) and by imposing the appropriate temporal constraints in the STN. Moreover, if tp is the start of a durative condition, we add the predicate to be maintained to the δ multiset, so that no effects violating the condition can be applied (because of line 7) until the time-point ending the durative condition is expanded, removing the condition from δ (line 11). Note that δ is a multiset so that if two durative conditions on the same predicate are active at the same time, we do not remove the condition upon the termination of the first interval. The temporal constraints added to the STN impose a total ordering (allowing contemporary nodes) on the expanded time-points (line 12) and "push" the non-expanded time-points to happen later with respect to the expanded time-point. This constraints on the future commitments can be either strict, (i.e. $>$) to impose a positive-time-separation between time-points that are mutex (e.g. between an effect and a supporting condition), or weak (i.e. $\geq$) to allow non-interfering effects or multiple conditions to happen at the same time. Note that the STN is checked upon expansion and, if found infeasible, the expansion is aborted by returning $\emptyset$, signaling that the expansion of tp is a dead-end.

The second kind of expansion is the "opening" of new action instances (algorithm 3), that is, we decide to start a new action. This choice adds to λ a new list of time-points ordered according to their constraints, representing the commitments on the future that the action brings. The computation of these commitments is analogous to the one described in algorithm 1 with the notable difference that START (resp. END) is interpreted against the time-point $t_{a\vdash}$ (resp. $t_{a\dashv}$)

Algorithm 2 Existing time-point expansion

```
1: procedure SUCCTP(s, tp)
2:     ⟨μ, δ, λ, χ, ω⟩ ← COPYSTATE(s)
3:     λ ← REMOVETP(λ, tp)                    ▷ This also pops empty lists from λ
4:     if kind(tp) ∈ {eff⁺, eff⁻} then
5:         if tp = ⟨eff⁺, p⟩ then μ ← μ ∪ {p}
6:         else if tp = ⟨eff⁻, p⟩ then μ ← μ/{p}
7:         if (⋃_{c∈δ} c) ⊈ μ then return ∅
8:     else if kind(tp) ∈ {cnd⊢, cnd⊣, cnd[]} then
9:         if c ∉ μ then return ∅
10:        if tp = ⟨cnd⊢, c⟩ then ADDTOMULTISET(δ, c)
11:        else if tp = ⟨cnd⊣, c⟩ then REMOVEONEFROMMULTISET(δ, c)
12:    PUSHTNCONSTRAINT(χ, tp ≥ ω)
13:    for all t ∈ l | l ∈ λ do
14:        if tp = ⟨eff⁺, p⟩ ∨ tp = ⟨eff⁻, p⟩ then
15:            if t = ⟨eff⁺, q⟩ ∨ t = ⟨eff⁻, q⟩ then
16:                if p ≠ q then PUSHTNCONSTRAINT(χ, t ≥ tp)
17:                else PUSHTNCONSTRAINT(χ, t > tp)
18:            else if kind(t) ∈ {cnd⊢, cnd⊣, cnd[]} then
19:                PUSHTNCONSTRAINT(χ, t > tp)
20:        else PUSHTNCONSTRAINT(χ, t ≥ tp)
21:    if ¬ CHECKTN(χ) then return ∅
22:    return {⟨μ, δ, λ, χ, tp⟩}
```

Algorithm 3 Action opening expansion

```
1: procedure SUCCACT(s, a)
2:     ⟨μ, δ, λ, χ, ω⟩ ← COPYSTATE(s)
3:     id ← MKFRESHINSTANCEID( )
4:     t_{a⊢}, t_{a⊣} ← ⟨act⊢, a, id⟩, ⟨act⊣, a, id⟩
5:     PUSHTNCONSTRAINT(χ, d_a^min ≤ t_{a⊣} − t_{a⊢} ≤ d_a^max)
6:     sl, el ← FILLTN(t_{a⊢}, t_{a⊣}, E_a^+, E_a^−, C_a, χ)
7:     λ ← λ + [ SORTBYACTIONTOTALORDER(sl ∪ el, a) ]
8:     if kind(ω) ∈ {eff⁺, eff⁻} then
9:         PUSHTNCONSTRAINT(χ, t_{a⊢} − ω > 0)
10:    else PUSHTNCONSTRAINT(χ, t_{a⊢} − ω ≥ 0)
11:    for all t ∈ λ do PUSHTNCONSTRAINT(χ, t − t_{a⊢} ≥ 0)
12:    if ¬ CHECKTN(χ) then return ∅
13:    return {⟨μ, δ, λ, χ, t_{a⊢}⟩}
```

representing the beginning (resp. the ending) of the action. Note that we assign a fresh instance id to $t_{a\vdash}$ and $t_{a\dashv}$ to maintain a correlation between an instance of an action starting and its ending; this will be exploited in order to reconstruct the plan to correlate time-points belonging to the same action instance. Also in this case, we check the consistency of the STN before confirming the successor to ensure the temporal feasibility of the action-opening choice.

A goal state $\langle \mu, \delta, \lambda, \chi, \omega \rangle$ for our search schema is a state where λ is empty, signaling that no commitments on the future are still to be achieved. Note that goals are automatically satisfied in such a state, because they are added to λ in the initial state (algorithm 1). Constructing a plan from a goal state can be done by simply extracting a consistent model from χ (such a model is guaranteed to exists because the STNs are kept consistent by the SUCCTP and SUCCACT successor functions) as follows. Let $\beta : \mathcal{T} \rightarrow \mathbb{Q}_{>=0}$ be a consistent model for χ, where $\mathcal{T}$ is the set of the time-points in χ. A solution plan encoded by β is as follows.

$$\{\langle \beta(s), a, \beta(e) - \beta(s) \rangle \mid s = \langle act^{\vdash}, a, i \rangle, e = \langle act^{\dashv}, a, i \rangle\}$$

Algorithm 4 Search algorithm

```
1: procedure SEARCH( )
2:     i ← GETINIT( );  g(i) ← 0
3:     Q ← NEWPRIORITYQUEUE( )
4:     PUSH(Q, i, h(i))
5:     while c ← POPMIN(Q) do
6:         if |c.λ| = 0 then return GETPLAN(c.χ)
7:         else
8:             for all s ∈ SUCC(s) do
9:                 g(s) ← g(c) + 1
10:                PUSH(Q, s, g(s) + h(s))
```

Intuitively, we take all the action startings (s) and the corresponding endings (e) and create an action instance in the plan that starts and lasts according to the STN model.

Computing subsumption of temporal states can be very hard (Coles and Coles 2016), we employ a best-first tree-search algorithm, using an A^*-like heuristic schema that sums the cost of the path to a state s (i.e. $g(s)$) with the heuristic estimation to reach the goal $h(s)$; in the next section we will detail how $h(s)$ is computed in our framework.

Theorem 3.1. *Let $\mathcal{P} \doteq \langle P, A, I, T, G \rangle$ be a planning problem with ICE, if $\mathcal{P}$ admits a solution plan, the search algorithm 4 terminates with a valid plan π.*

As a final note, it is possible to instrument the search to consider states that have the same μ and the same λ sizing as identical, without losing soundness. This transforms the search state from an infinite tree to a finite graph improving the planning performance, but loosing the guarantee to eventually find a plan if it exists. This under-approximation of the problem is used as a pre-processing step in our implementation: if the solving on the finite graph fails, we resort to the full-blown tree search.

Heuristic and Relaxation. To guide our search, we define a relaxation of the input temporal planning model into a classical planning model, allowing the use of any domain-independent heuristic designed for classical planning in our context. The overall idea behind our relaxation is to create a classical planning action for each time-point in the planning problem with ICE, impose ordering constraints among time-points that we know are ordered, and use such a relaxation to compute heuristic values, by translating an arbitrary search state for the original ICE planning problem into a classical state for the relaxation.

Our relaxation is defined as a STRIPS classical planning problem on a set of predicates P' defined as $P \cup \{q_i^a \mid i \in \{0, \cdots, |ttp(a)|\}, a \in A\} \cup \{g_x \mid x \in ttp(T, G)\}$, having initial state $I \cup \{q_0^a \mid a \in A\}$. We create a classical planning action for each time-point of each action in A, one for each TIL in T and one for each goal in G. Algorithm 5 formalizes the construction of the set of relaxed classical actions A': we characterize each action $a' \in A'$ with its precondition ($pre_{a'}$), add and delete effects ($eff^+_{a'}$ and $eff^-_{a'}$). The additional predicates q_i^a are used to impose precedences between time-points belonging to the same action. Initially each q_0^a is true, signaling that no action is started; when an action starts executing its time-points, the "true value" is moved from q_i^a to q_{i+1}^a (i.e. q_i^a is set to false and q_{i+1}^a to true), until the last

Algorithm 5 Relaxed actions

procedure GETRELAXATIONACTIONS( )
  for all $a \in A$ **do**
    $i \leftarrow 0$
    for all $\langle x, p\rangle \in$ SORTBYACTIONTOTALORDER($ttp(a)$) **do**
      $c, e^+, e^- \leftarrow \emptyset, \emptyset, \emptyset$
      $j \leftarrow (i+1) \bmod |ttp(a)|$
      if $x = eff^+$ **then** $e^+ \leftarrow \{p\}$
      else if $x = eff^-$ **then** $e^- \leftarrow \{p\}$
      else $c \leftarrow \{p\}$
      $pre_{a'}, eff^+_{a'}, eff^-_{a'} \leftarrow \{q^a_i\} \cup c, \{q^a_j\} \cup e^+, \{q^a_i\} \cup e^-$
      $A' \leftarrow A' \cup a'$; $i \leftarrow j$
  for all $\langle x, p\rangle \in ttp(T, G)$ **do**
    $c, e^+, e^- \leftarrow \emptyset, \emptyset, \emptyset$
    if $x = eff^+$ **then** $e^+ \leftarrow \{p\}$
    else if $x = eff^-$ **then** $e^- \leftarrow \{p\}$
    else $c \leftarrow \{p\}$
    $pre_{a'}, eff^+_{a'}, eff^-_{a'} \leftarrow c, \{g_{\langle x,p\rangle}\} \cup e^+, e^-$
    $A' \leftarrow A' \cup a'$
  return A'

time-point is executed, at which point we reset q^a_0 to true. This is in fact a unary counter over the time-points of a durative action. The additional predicates g_x, instead, are used as the relaxation goals together with the q^a_0 predicates for each $a \in A$ (i.e. $G' \doteq \{g_x \mid g_x \in P\} \cup \{q^a_0 \mid a \in A\}$). The g_x predicates are set to true by relaxed actions that represent a goal or a TIL, imposing to the relaxation that all the goals and TILs need to be encountered in a plan. For the sake of simplicity, we do not impose precedences among actions representing goals or TILs because these are not guaranteed to be totally-ordered.

From the relaxed model, we can compute heuristic values as follows. A given search state $s \doteq \langle \mu, \delta, \lambda, \chi, \omega\rangle$ corresponds to a state s' defined as $\mu \cup \{q^a_0 \mid a \in A\} \cup \{q^a_{|ttp(a)|-|l|} \mid l \in \lambda, \langle act^\dashv, a, i\rangle \in l\} \cup \{g_x \mid x \notin \lambda \wedge x \in ttp(T, G)\}$ in the relaxed model. Note that q^a_0 is always kept to true to allow self-overlapping (Rintanen 2007) of actions. If this feature is not needed, one can remove the $\{q^a_0 \mid a \in A\}$ elements from s'.

We can compute the heuristic value for any state s as $h(s')$, being h any classical planning heuristic.

Simultaneity Optimization. The search schema above can be optimized by "compressing" multiple time-points that happen at the same time in a single search step. In particular, it is possible to recognize multiple conditions and multiple effects that must happen at the same time and force the search to expand these nodes in a static, fixed order without intermediate branching. We syntactically recognize three patterns of such *simultaneous* time-points. The first are the effect time-points (independently of whether they are add or delete) belonging to the same action that are scheduled at the same time (e.g two effects, both at START $+ k$). The second are condition time-points ($cnd^\vdash$, $cnd^\dashv$, or $cnd^{[]}$) belonging to the same action that are scheduled at the same time (e.g two conditions, both at START$+k$). The third are time-points of heterogeneous type (i.e. a condition and an effect) belonging to the same action that are scheduled at the same time. For this last third case, we need to be careful in checking that no other pair of condition and effect belonging to a different action would require to be executed simultaneously to the pair that we are compressing.

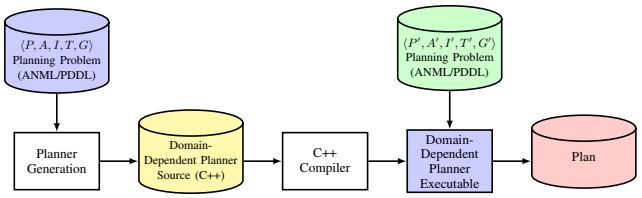

Figure 1: The compilation data-flow.

4 Quasi-Domain-Dependent Planner Generation

One of the major areas in which planning problems with ICE arise is the orchestration of flexible production. In fact, it is not uncommon to have some operation requiring a certain amount of time that must immediately (or within some deadline) be followed by other operations. Being this the intended deployment setting of our planner, we recognize the existence of a "deployment phase" that is a moment in which an instance of the planner is given a certain domain and it will be required to solve lots of different problems on that domain. For this reason, we equipped our planner with the capability of generating compilable source code that embeds the search strategy described in the previous section. In this way, we can greatly simplify the computational work of the planner implementation by "hard-coding" and optimizing parts of the problem such as the number of objects and the actions, obtaining performances that get closer to domain-dependent planners implementations. We call this compilation "quasi-domain-independent planner generation".

The data-flow of the compilation is shown in figure 1. Starting from a planning problem instance $\langle P, A, I, T, G\rangle$, the generator produces a self-contained C++ source code implementing the heuristic search described in the previous section, fixing both the number of objects and the set of actions of the problem. All the parts of the approach benefit from this assumption: the state is implemented as a class that has one field for each predicate and each time-point is given a unique ID to simplify memory management and transforming all the needed maps into arrays. The program obtained by compiling the generated source code can solve the original problem or any other problem $\langle P', A', I', T', G'\rangle$ having a set of objects[2] that is a subset of the objects of the original problem. In fact, we allow the user to override the initial state and the goal of the problem as well as to disable some of the objects in the original problem.

In situations where we have a family of problems having the same structure and differing only because of the number of objects, initial state and goals, we can use this compilation by creating a planning problem that has at least as many

[2]While the formalization in the previous section assumed (as customary) a ground model, the code generation as well as the implementation accept a lifted model, where a certain number of object instances can be used to parameterize actions and predicates.

objects as the biggest instance and create a compiled executable with our technique. The very same executable can then be used to solve all the instances in the batch efficiently. Note that in this schema, the cost of compilation (that is non-negligible for complex problem instances) is payed only once for the entire set of instances sharing the same structure. This is particularly useful in flexible production deployment, because domain experts can safely estimate the maximum number of items, machines and personnel for a specific factory/deployment and we can thus use this information to compile (only once) an efficient quasi-domain-dependent planner. Finally, note that our implementation is still capable of symbolic planning (i.e. planning without using the intermediate compilation step), granting a safe fallback if for some reason one instance is larger or different from the one used for compilation.

5 Related Work

Intermediate conditions and effects have been recognized as an important feature for temporal planning in several works. (Cimatti et al. 2018) indicates a practical case for ICE: the authors propose a compilation from planning problems having temporal uncertainty in the duration of PDDL 2.1 actions that produces plain temporal planning problems with ICE.

In his commentary to the PDDL 2.1 language, Smith (2003) recognizes how the limitations imposed by the language make it "exceptionally cumbersome" to encode ICE. Another instance of compilation of ICE into PDDL 2.1 can be found in (Fox, Long, and Halsey 2004) where the authors explain the so-called "clip-action" construction that can be used to "break" a durative actions having ICE into a number of smaller durative actions without ICE at the cost of adding required concurrency (Cushing et al. 2007), additional fluents and additional (clip) actions. These compilations can be used to transform a planning problem with ICE into a PDDL 2.1 planning problem[3], making it possible to use any state of the art temporal planner for PDDL 2.1 to solve problems with ICE. The current best planners for PDDL 2.1 are based on heuristic search (e.g. (Coles et al. 2010; Eyerich, Mattmüller, and Röger 2012)); in this paper we build on these approaches and propose a technique that natively supports and reasons on ICE. Our experiments indicate that this approach is far superior to PDDL compilations.

The ANML planning language (Smith, Frank, and Cushing 2008) directly and explicitly supports ICE, in fact our planner, TAMER, is based on this language. Unfortunately, few planners currently support ANML and none of them implements a heuristic-search approach. In fact, FAPE (Dvorak et al. 2014) implements a plan-space search focusing on hierarchical task decomposition (but still supports ICE with an integer time interpretation), while LCP (Bit-Monnot 2018) implements an encoding of the bounded planning problem into a constraint satisfaction problem.

A recent publication (Micheli and Scala 2019) proposed the use of temporal metric trajectory constraints to encode the temporal features of planning problems. The authors focus on a language where all actions are instantaneous, and ICE can be expressed by creating an action for each timepoint and imposing appropriate trajectory constraints. In this paper, we retain the "actions-as-intervals" idea adopted by both PDDL 2.1 and ANML, and experimentally show that this can give advantages on some domains (in particular on one proposed by Micheli and Scala).

Finally, we complement our contribution by presenting and evaluating a compilation of our planner technique into executable code. We borrowed this idea from the verification community, in particular from the SPIN model-checker (Holzmann 1997). In planning, this idea has been recently exploited by Francès et al. (2017) where the planning actions are seen as opaque simulators changing the planning state. The idea evolved in the IW planner (Lipovetzky and Geffner 2017) that can solve black-box classical planning problems by incrementally bounding the "width" of the problem. We highlight that all these approaches are designed and work for classical planning, while here we tackle temporal planning. Moreover, we do retain domain-independent heuristic computation even in the compiled executable as we start from a "white-box" model of the system.

6 Experimental Evaluation

We experimentally evaluate the merits of both our search schema and the impact of the code generation on a comprehensive set of benchmarks, similar to the one used in (Micheli and Scala 2019). In particular, we took all the MAJSP, Temporal IPC and Uncertainty IPC instances (we disregarded the "HSP" instances that are not directly expressible in a planning problem with ICE). The MAJSP domain is a job-shop scheduling problem in which a fleet of moving agents transport items and products between operating machines. We took all the 240 instances of this domain and we manually re-coded them in ANML using ICE. The Temporal IPC class is composed of temporal planning domains (without ICE) of the IPC-14 competition (Vallati et al. 2015) for a total of 98 planning instances. The Uncertainty IPC class consists of the same planning instances where the durations of some actions are assumed to be uncontrollable and the rewriting in (Cimatti et al. 2018) is used to produce equivalent temporal planning problems with ICE. Finally, we added a new domain, called PAINTER: a worker has to apply several coats of paint on a set of items guaranteeing a minimum and a maximum time between two subsequent coats on the same item. We created 300 instances of this domain by scaling the number of coats (from 2 to 11) and items (from 1 to 30) and formulated each instance in both ANML, TPP (the language of TPACK) and PDDL 2.1.

We implemented the heuristic search technique of section 3 in a planner called TAMER. Our planner is written in C++ and accepts both PDDL 2.1 and ANML specifications as input (recall that ANML allows ICE, while PDDL 2.1 does not). TAMER uses the standard h_{add} classical planning heuristic (Bonet and Geffner 2001) on the relaxation defined in section 3. The quasi-domain-dependent generator of section 4 is implemented in C++ and uses the GCC C++ compiler to produce executables from the generated code.

[3]Formally, this is only possible if we assume a ϵ-separation semantics, but we disregard this detail for the rest of the paper.

Temporal IPC (a)

Domain (# inst.)	ITSAT	OPTIC	TFLAP	TPACK	ANML SMT	TAMER
DRIVERLOG (20)	**18**	15	17	14	4	12
FLOORTILE (8)	**8**	7	5	4	0	7
MAPANALYSER (20)	15	0	15	**20**	0	8
MATCHCELLAR (10)	**10**	9	**10**	6	6	7
SATELLITE (20)	**20**	14	**20**	9	3	9
TMS (20)	**20**	1	0	1	0	0
Total (98)	**91**	46	67	54	13	43

Uncertainty IPC (b)

Domain (# inst.)	ITSAT clip*	ITSAT cont.*	OPTIC clip	OPTIC cont.	TFLAP clip	TFLAP cont.	TPACK	ANML SMT	TAMER
DRIVERLOG (20)	0	0	0	5	0	0	**7**	4	5
FLOORTILE (8)	0	0	0	0	0	0	**4**	0	**4**
MAPANALYSER (20)	0	0	0	0	0	0	**2**	0	1
MATCHCELLAR (10)	4	0	3	**10**	0	0	5	5	5
SATELLITE (20)	1	0	0	1	0	0	**5**	1	1
TMS (20)	**1**	0	0	0	0	0	0	0	0
Total (98)	7	0	3	16	0	0	**23**	10	16

PAINTER (c)

#Coats	OPTIC clip	OPTIC cont.	TFLAP clip	TFLAP cont.	TPACK	FAPE	ANML SMT	TAMER
2	0	3	0	0	2	2	1	**30**
3	0	3	0	0	2	2	0	**30**
4	3	4	0	0	7	11	6	**30**
5	1	4	0	0	5	4	4	**30**
6	0	4	0	0	4	3	3	**30**
7	0	4	0	0	3	3	2	**30**
8	0	3	0	0	2	2	2	**30**
9	0	3	0	0	2	2	2	**30**
10	0	3	0	0	2	2	2	**30**
11	0	3	0	0	2	2	1	**30**
Tot.	4	34	0	0	31	33	23	**300**

MAJSP (d)

#Jobs	OPTIC clip	OPTIC cont.	TFLAP clip	TFLAP cont.	TPACK	FAPE	ANML SMT	TAMER
1	14	0	0	23	56	NA	**60**	56
2	0	0	0	2	45	NA	45	**52**
3	0	0	0	0	27	NA	30	**46**
4	0	0	0	0	14	NA	17	**43**
Tot.	14	0	0	25	142	NA	152	**197**

Figure 2: Coverage results for IPC-14 (a-b) and for PAINTER and MAJSP (c-d).

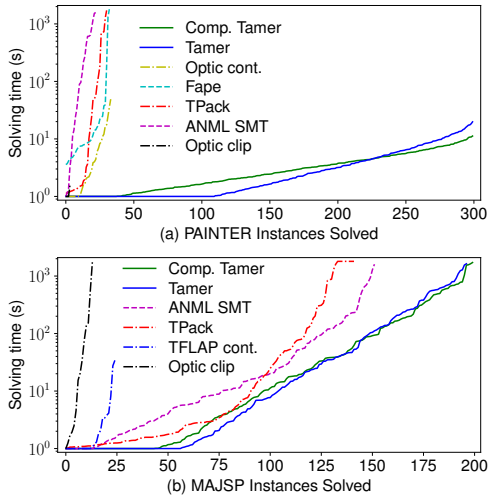

Figure 3: Cactus plots for the industrial domains.

In our experimental analysis we compare against a number of planners. We use ITSAT (Rankooh and Ghassemi-Sani 2015), OPTIC (Benton, Coles, and Coles 2012) and TFLAP (Sapena, Marzal, and Onaindia 2018) as representatives of PDDL 2.1 planners: since PDDL 2.1 does not natively support ICE, we employ the clip-construction described in (Fox, Long, and Halsey 2004) and the container-construction described in (Smith 2003) to capture the ICE semantics. To use ITSAT, we re-scaled the actions durations to integer values in the Uncertainty IPC class (this is remarked by an asterisk in the tables). We did the same for the PAINTER and MAJSP domains, but ITSAT always crashed. We also consider two ANML planners, namely FAPE (Dvorak et al. 2014) and ANML SMT; the latter is an adaptation of the encoding described in (Shin and Davis 2005) for the ANML language[4]. We could not use FAPE for MAJSP because FAPE does not support numerical fluents nor for the IPC domains because FAPE, differently from ANML SMT, is unable to parse PDDL 2.1. Finally, we include the TPACK planner (Micheli and Scala 2019) in our experiments, by manually re-coding all the new instances in the TPACK input language. To evaluate the performance of the code-generation optimization, we compiled the (automatically-generated) quasi-domain-dependent planner for the largest instance of both PAINTER and MAJSP, and used the resulting executable to solve all the instances for that domain. The GCC compilation times for such domains are 203 and 492 seconds, respectively. We indicate this approach as COMP. TAMER. We ran all the experiments on a Xeon E5-2620 2.10GHz with 1800s/15GB time/memory limits. TAMER and all the benchmarks are available at: https://es-static.fbk.eu/people/amicheli/resources/aaai20.

[4]We implemented ANML SMT using the MathSAT5 (Cimatti et al. 2013) solver. The encoding is similar to (Bit-Monnot 2018).

Results. The coverage results for our experiments are presented in figure 2, while figure 3 depicts the time performance on the PAINTER and MAJSP domains.

The Temporal IPC domains show how, on domains without ICE, our planner exhibits comparable, but inferior, performance w.r.t. OPTIC and TPACK. This is expected, as our search schema introduces some overhead in order to support ICE. Also, IPC domains are very challenging from a "classical planning" point of view, but are not very rich in terms of temporal features. This fact is reflected also in figure 2b, where our planner is able to solve the second highest number of instances behind TPACK (OPTIC achieves the same coverage, mostly due to MatchCellar, where it excels). In fact, the Uncertainty IPC instances have very simple and localized ICE, and the propositional complexity is predominant.

Figure 2b shows that TAMER outperforms all the competitors in all the PAINTER instances. In fact, we tried to further scale the same domain (up to 32 coats and 30 items) and TAMER is still able to solve all the instances. We highlight how all the competitors (excepting Optic with the clip-action construction) solve almost the same number of instances, that are an order of magnitude less that those solved by TAMER. This is also evident from the cactus plot in figure 3a

that also clearly depicts how the generated quasi-domain-dependent planner (produced from the largest ANML model by our approach) surpasses the performance of the symbolic search as the size of the instances grows. In this schema, the GCC compilation cost is payed only once and the resulting solver is used uniformly on all the instances (the cactus plots do not consider the compilation times reported in the text above). The advantage of the generated quasi-domain-dependent planner is vast: note that the time scale of the plot is logarithmic and that the quasi-domain-dependent planner cuts the run-time in half for the harder instances. The situation is similar in the MAJSP domain. We highlight that these instances are taken from the TPACK distribution and both TAMER and ANML SMT are able to outperform TPACK. In this case, the impact of the code generation is less dramatic, but COMP. TAMER is able to solve 200 instances in total, beating any other approach.

Finally, we remark that our code generation procedure can be invoked on each problem instance, producing executables that are tailored to a specific set of objects but allow the change of the initial state and/or the goal. The GCC compilation time is non-negligible for larger instances, but the run-time is always smaller than TAMER: the average run-time speedup is 243%. Concerning coverage, we managed to solve 205 instances of MAJSP, taking into account the compilation time for the timeout.

7 Conclusions

In this paper, we presented a novel heuristic-search technique for solving planning problems exhibiting intermediate conditions and effects (ICE), that are useful to naturally model and effectively solve practical and industrial problems. The technique is complemented by code generation capabilities that further push the performance of the solver.

Future work includes the exploration of different heuristics on our relaxation of the problem and the definition of alternative relaxations. Moreover, considering optimality for problems with ICE is another interesting research line.

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS 2012*.

Bit-Monnot, A. 2018. A constraint-based encoding for domain-independent temporal planning. In *CP 2018*, 30–46.

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

Cimatti, A.; Griggio, A.; Schaafsma, B.; and Sebastiani, R. 2013. The MathSAT5 SMT Solver. In *TACAS 2013*.

Cimatti, A.; Do, M.; Micheli, A.; Roveri, M.; and Smith, D. E. 2018. Strong temporal planning with uncontrollable durations. *Artificial Intelligence*.

Coles, A., and Coles, A. 2016. Have I been here before? state memoization in temporal planning. In *ICAPS 2016*.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *ICAPS 2010*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *IJCAI 2007*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence*.

Dvorak, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and acting with temporal and hierarchical decomposition models. In *ICTAI 2014*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2012. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Towards Service Robots for Everyday Environments - Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*.

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of pddl2. 1. In *ECAI 2004*.

Francès, G.; Ramírez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely declarative action descriptions are overrated: Classical planning with simulators. In *IJCAI 2017*, 4294–4301.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*.

Holzmann, G. J. 1997. The model checker SPIN. *IEEE Transactions of Software Engeenering* 23(5):279–295.

Lipovetzky, N., and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *AAAI 2017*, 3590–3596.

Micheli, A., and Scala, E. 2019. Temporal planning with temporal metric trajectory constraints. In *AAAI 2019*, 7675–7682.

Rankooh, M. F., and Ghassem-Sani, G. 2015. Itsat: an efficient sat-based temporal planner. *Journal of Artificial Intelligence Research*.

Rintanen, J. 2007. Complexity of concurrent temporal planning. In *ICAPS*.

Sapena, O.; Marzal, E.; and Onaindia, E. 2018. TFLAP: a temporal forward partial-order planner. Technical report, 2018 International Planning Competition, Temporal Tack. https://ipc2018-temporal.bitbucket.io/planner-abstracts/team2.pdf.

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a sat-based planner. *Artificial Intelligence*.

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

Smith, D. E. 2003. The case for durative actions: A commentary on pddl2. 1. *Journal of Artificial Intelligence Research*.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine*.