



Heriot-Watt University  
Research Gateway

# Temporal Planning with Incomplete Knowledge and Perceptual Information

**Citation for published version:**

Carreno, Y, Petillot, Y & Petrick, RPA 2022, 'Temporal Planning with Incomplete Knowledge and Perceptual Information', *Electronic Proceedings in Theoretical Computer Science*, vol. 362, pp. 37-53.  
<https://doi.org/10.4204/EPTCS.362.6>

**Digital Object Identifier (DOI):**

[10.4204/EPTCS.362.6](https://doi.org/10.4204/EPTCS.362.6)

**Link:**

[Link to publication record in Heriot-Watt Research Portal](#)

**Document Version:**

Publisher's PDF, also known as Version of record

**Published In:**

Electronic Proceedings in Theoretical Computer Science

**Publisher Rights Statement:**

© Y. Carreno, Y. Petillot, R. Petrick.

**General rights**

Copyright for the publications made accessible via Heriot-Watt Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy**

Heriot-Watt University has made every reasonable effort to ensure that the content in Heriot-Watt Research Portal complies with UK legislation. If you believe that the public display of this file breaches copyright please contact [open.access@hw.ac.uk](mailto:open.access@hw.ac.uk) providing details, and we will remove access to the work immediately and investigate your claim.

# Temporal Planning with Incomplete Knowledge and Perceptual Information

Yaniel Carreno\* Yvan Petillot Ronald P. A. Petrick

Edinburgh Centre for Robotics, Edinburgh, United Kingdom

Heriot-Watt University and The University of Edinburgh, Edinburgh, United Kingdom

{y.carreno, y.r.petillot, r.petrick}@hw.ac.uk

In real-world applications, the ability to reason about incomplete knowledge, sensing, temporal notions, and numeric constraints is vital. While several AI planners are capable of dealing with some of these requirements, they are mostly limited to problems with specific types of constraints. This paper presents a new planning approach that combines contingent plan construction within a temporal planning framework, offering solutions that consider numeric constraints and incomplete knowledge. We propose a small extension to the Planning Domain Definition Language (PDDL) to model (i) incomplete, (ii) knowledge sensing actions that operate over unknown propositions, and (iii) possible outcomes from non-deterministic sensing effects. We also introduce a new set of planning domains to evaluate our solver, which has shown good performance on a variety of problems.

## 1 Introduction and Motivation

Automated planning is widely used as a tool for autonomous agents to achieve their mission goals. In the past decade, AI planners have been introduced in a large number of real-world applications [30] to deal with the challenges that arise in these scenarios, such as the incompleteness of the domain definition and the complexity of dynamic models that capture numeric and temporal constraints. Hybrid temporal planners are able to reason with both discrete and continuous numeric changes over time to generate realistic action schedules (plans) capable of supporting concurrent actions, synchronisation of multiple tasks, and deadlines. Several solutions to planning with continuous and discrete effects [28, 32, 16] base their reasoning on deterministic models. These solvers generate plans by scheduling sequences of actions that satisfy numeric and temporal constraints. However, their applicability is limited when parts of the domain are incomplete or unknown. Contingent planning [38, 43] copes with certain types of incomplete information by treating the plan as a decision tree with different contingent branches that could arise. A contingent plan guides the agent to act conditionally to achieve the goal, with actions in the decision tree enabling the planner to decide which branch to take.

In this work, we focus on temporal planning with numeric constraints where the action sequences required to reach the goals give rise to conditional plans. For instance, consider the following example:

**Example 1.** (*Valve Manipulation*) An offshore scenario (Figure 1 (left)) includes a set of blowout preventers (BOPs) which are controlled by manipulating valves. An autonomous underwater vehicle (AUV) must close two valves ( $v1$  and  $v2$ ) during a mission, and must record and communicate data every time it manipulates a valve. The AUV starts at the deployment base. From there, it can navigate to the BOPs and manipulate a valve. The AUV's actions will depend on the valve state: if a valve is open, then it should be closed; if a valve is closed, then no action is needed. The valve state can be checked using a

---

\*Contact Author.

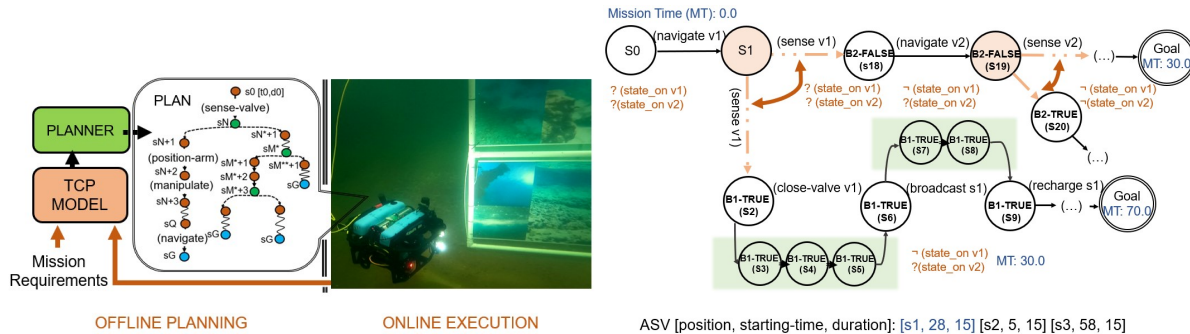


Figure 1: **(left)** Example of the planning and execution process required to solve a problem in Example 1. **(right)** Graphical representation of the plan solution for Example 1.

*sensing action. The AUV may also need to recharge using an autonomous surface vehicle (ASV) which is deployed in different recharge points at different times.*

Figure 1 (right) shows a plan solution for this problem. We note that no sequence of actions allows the AUV to achieve the goal without first determining the state of the valves. As a result, choosing the correct actions to execute depends on the outcome of sensing the valve states (open or closed). Numeric constraints are required to control the data recorded when a valve is manipulated. Therefore, if the valve is open, the AUV should execute a manipulation action and navigate to the surface to report the action’s implementation. Temporal constraints are also essential for scheduling recharge activities due to the ASV’s time availability at different locations. Different sensing outcomes (red) can lead to different task sequences with extra actions (green) requiring more energy and time to complete the mission.

In this paper, we formalise temporally-contingent planning (TCP) problems, which describe a partially observable environment with sensing actions (PPOS) and time constraints, and introduce a new compilation-based planning approach called Temporal Contingent Effect (TraCE) that combines temporal plan construction with contingent planning. TraCE analyses the PPOS as a Fully-Observable Non-Deterministic (FOND) planning problem using a modified version of the approach in [5] to convert PPOS into FOND problems that include temporal constraints. Both deterministic (physical) actions and non-deterministic (sensing) actions are treated as durative actions. Plan branches are computed by considering the branching points, arising from sensing actions, and branch depth. The plans generated by TraCE are trees of actions (Figure 1 (right)), where the actions in a branch satisfy the ordinary propositional pre-conditions but are scheduled by a sub-solver that also considers temporal and numeric requirements. The approach is evaluated on a set of planning problems that include real robotic applications.

## 2 Related Work

Many temporal planning models introduce an explicit notion of time [24]. Examples include solvers dealing with temporal coordination [20], continuous effect requirements [16], and preferences [2]. Such approaches are capable of solving a large number of well-established planning domains [31], and real-world domains [9, 8, 12]. Our framework aims to solve a new set of problems that require reasoning about incomplete information in addition to the temporal and numeric requirements. In this work, we focus on offline planning strategies. Previous examples of offline contingent planning solvers include MBP [4], Contingent-FF [29], and POND [6]. Although these planners can solve large problems at different hardness levels, they do not scale well, mostly due to the underlying belief state representation.

DNFct [41] and CLG [1] (offline version) outperform previous approaches, finding solutions in a shorter time with better scalability. A different contingent planning approach is PKS [39, 40] which attempts to model the knowledge state directly. Amongst the group of planners that deal with FOND planning problems, PO-PRP [34] extends PRP [35] to compute strong cycle plans. ProbPRP [7] is an extension of PO-PRP that computes policies to overcome deadends. HCP-ASP [44] and HCPlan [36] demonstrate the applicability of contingent planning in hybrid approaches. Our work inherits some of the ideas introduced by the prior approaches regarding implementing hybrid solutions and the translation (or compilation) of nondeterministic problems for planning.

Few approaches in the literature combine temporal and contingent planning to solve nondeterministic temporal planning problems. CTP [42] allows the construction of conditional plans with temporal constraints. Strategies based on Simple Temporal Networks with Uncertainty (STNU) such as [14] are used as temporal scheduling tools where conditions and decisions can be added to STNUs [17, 45]. [18] present an encoding to Conditional Simple Temporal Networks with Uncertainty and Resources (CST-NURs) with promising results in a set of applications that model temporal and numeric constraints. A solution that considers planning and meeting temporal problem constraints is the TCP framework [21], which includes time and contingency notions in the model. This framework differs from the contingent planning problems we are interested in, where branch creation leads to solve the incomplete knowledge. Some work associated with temporal plan merging [25, 10] and opportunistic planning consider temporal [13] and resource [15] constraints to generate branches in a temporal plan. [11] presents an approach to solve TCP problems considering the work implemented in [37] that translates contingent planning into classical problems. TraCE differs from this approach in the problem translation and the tree expansion.

### 3 Problem Formalisation

We begin by formulating the TCP problem. We use PDDL2.1 [22] which is typically used for temporal planning problems:

**Definition 1.** A temporal planning problem is a tuple  $P_T := \langle P, V, A_T, I_T, G_T, T \rangle$ , where  $P$  is a set of Boolean variables;  $V$  is a vector of real variables (numeric fluents);  $A_T$  is a set of instantaneous and durative actions, where the duration of the actions is controllable and known;  $I_T$  is a function  $I_T : P \cup V \rightarrow \{\top, \perp\} \cup \mathbb{R}$  describing the initial state;  $G_T$  is a set of goals, where goals in  $G_T$  are Boolean variables from  $P$  or inequalities over real variables in  $V$  that represent the objectives that must be achieved;  $T$  is a set of timed initial literals (TILs).

A timed initial literal (TIL) [19] is a pair  $(t, p)$  where  $t$  is the rational-valued time of occurrence of a Boolean variable  $p$ , where  $p \in P$  and  $p \rightarrow \{p, \neg p\}$ . A TIL  $(t, p)$  defines the time  $t$  that the Boolean variable  $p$  becomes true ( $p$ ), and a TIL  $(t, \neg p)$  describes the time  $t$  that  $p$  becomes false ( $\neg p$ ).

**Definition 2.** A durative action  $a_d \in A_T$  is a tuple of the form  $\langle a_{d_{pre}}, a_{d_{eff}}, a_{d_{dur}} \rangle$ .  $a_{d_{pre}}$  is a set of conditions of the following types that must hold for the action to be applicable: at-start ( $a_{d_{pre}^-}$ ), over-all ( $a_{d_{pre} \leftrightarrow}$ ), and at-end ( $a_{d_{pre}^+}$ ).  $a_{d_{eff}}$  is the set of action effects of the types: positive starting effects ( $a_{d_{eff}^{++}}$ ), negative starting effects ( $a_{d_{eff}^{--}}$ ), numeric starting effects ( $a_{d_{eff}^{+n}}$ ), continuous numeric effects ( $a_{d_{eff}^{+n}}$ ), positive ending effects ( $a_{d_{eff}^{-+}}$ ), negative ending effects ( $a_{d_{eff}^{-}}$ ), and numeric ending effects ( $a_{d_{eff}^{-n}}$ ).  $a_{d_{dur}}$  is a set of action durations.

A solution to a TCP problem  $P_T$  is a time-aware plan  $\Pi_T = \{a_1 \cdots, a_n\}$  described by a sequence of durative and instantaneous actions, where each action  $a_i$  is applicable, and  $\Pi_T$  achieves the goals in  $G_T$  satisfying the temporal constraints. For instance, each branch in Figure 1 (right) from the initial state  $I_T$

to the goal  $G_T$  is a single sequence of durative actions. Note that such plans do not consider incomplete information or the effects of sensing actions.

We now consider contingent planning problems which include partial observability and non-deterministic (sensing) actions in the model, but no temporal constraints.

**Definition 3.** A contingent planning problem is a tuple  $P_C := \langle P, A_C, O, I_C, G_C \rangle$ , where  $P$  is set of Boolean variables describing the state of the world;  $A_C$  is a set of physical actions;  $O$  is a set of sensing actions, separate from  $A_C$  such that  $O \cap A_C = \emptyset$ ;  $I_C \subseteq P$ , is the set of clauses over  $P$  that denotes the initial state; and  $G_C$  is a set of propositions over  $P$  representing the goal condition.

A proposition  $p \in P$  or its negation  $\neg p$  defines a literal  $l \in L$ , where  $L$  is a set of literals and  $\bar{L} = \{\neg l \mid l \in L\}$ . A set  $L'$  of literals is consistent if the condition  $\{p, \neg p\} \not\subseteq L'$  holds for every  $p$ ; and complete if the condition  $\{p, \neg p\} \cap L' \neq \emptyset$  holds for every  $p \in P$ . The complement of a literal  $l$  is defined as  $\bar{l}$  such that  $\bar{\bar{l}} = l$  and  $p = \neg \bar{p}$  for  $p \in P$ .

A state  $s$  is defined as a consistent and complete set  $L$  of literals. A state  $s$  satisfies a conjunction of literals  $L''$  ( $s \models L''$ ) if  $L'' \subseteq s$ . A state  $s$  satisfies a literal  $l$  ( $s \models l$ ) if  $l \in s$ . A belief state  $S$  is a set of states. The belief state  $S \models l$  if  $s \models l$  for every  $s \in S$ . Finally,  $S \models L''$  if  $s \models L''$  for every  $s \in S$ .

The set of states in the belief state  $S$  capture the alternative ways that the world could be configured. In a contingent planning model, a literal as being “unknown” if it appears in more than one state in a belief state with different mappings. So, if  $s \models l$  is true in each state  $s$ , then  $s$  is said to be known. However, if there exist states  $s_1$  and  $s_2$  such that  $s_1 \models l$  and  $s_2 \models \neg l$  then  $l$  would be considered unknown since either value of  $l$  is considered possible. For instance, in Example 1 a belief state  $S$  includes the possible outcomes of the `(state_on v1)` variable which are `(state_on v1)` and `¬(state_on v1)`.

Actions are defined as follows:

**Definition 4.** A physical action  $a \in A_C$  is a tuple  $\langle a_{pre}, a_{eff} \rangle$ , where  $a_{pre}$  is a set of atomic propositions indicating the preconditions to implement  $a$ , and  $a_{eff}$  represents a set of action outcomes.  $a_{eff}$  describes a set of pairs  $\langle c, l \rangle$ , which capture its conditional effects, where  $c$  is a set of possible effects defined by a set of literals  $L$  and  $l$  is a literal.

**Definition 5.** A sensing action  $o \in O$  is a tuple  $\langle o_{pre}, o_{eff} \rangle$ , where  $o_{pre}$  is a set of atomic propositions indicating the preconditions to implement  $o$ , and  $o_{eff}$  completely characterises an unknown literal  $l$ , by considering the possible outcomes for a given proposition. Therefore,  $o_{eff}$  includes a set of possible effects  $o'_{eff}$  and  $o''_{eff}$  that uncover the value of  $l$ .

This is for a belief state  $S$  a sensing action effects define two independent belief state  $S^+$  and  $S^-$  that enclose the states where  $l$  is possible, and states where  $\neg l$  is possible, respectively. Considering Definition 5, sensing actions are nondeterministic. Sensing actions lead to a possible set of outcomes (states), enclosed in the belief state, associated with the truth value of a literal. For a sensing action the belief state  $S_o$  gives rise to two possible belief states, depending on the value of the underlying literal ( $l$  or  $\neg l$ ). This makes the model nondeterministic at plan time but the belief state is always certain in each case. The possible belief states associated with a sensing action are:  $S_o^+ = \{s \mid s \in S, s \models l(o)\}$  and  $S_o^- = \{s \mid s \in S, s \models \bar{l}(o)\}$ .

A solution to  $\Pi_C$  is a contingent-aware plan  $P_C$  with a branching structure induced by the sensing outcomes. For instance, the plan in Figure 1 (right) is a solution to a contingent planning problem, assuming the recharge and broadcast actions, which include temporal and numeric constraints, are ignored.

### 3.1 Temporally-Contingent Planning Problem

A TCP problem is a special case of the PPOS problem where the source of uncertainty is connected to the outcome of the sensing actions [34], and temporal and numeric notions are introduced in the model. We

consider simple PPOS problems [5] which can be mapped into FOND planning problems [34]. In particular, we assume that (i) non-unary clauses in the initial state  $I$  are invariant, which are states that hold in every world state [26], (ii) hidden literals do not emerge in the effects of a non-deterministic action, and (iii) uncertainty decreases monotonically, i.e., unknown properties cannot become unknown again after becoming known. As a result, the TCP problem is a FOND problem with temporal requirements and real variables.

**Definition 6.** A temporally-contingent planning problem is a tuple  $P_{TC} := \langle P, V, A, \Delta, I, G, T \rangle$ , where  $P$  is set of Boolean variables describing the state of the world;  $V$  is a vector of real variables (numeric fluents),  $A$  is a set of durative physical (deterministic) actions;  $\Delta$  is a set of durative sensing (nondeterministic) actions, separate from  $A$  such that  $\Delta \cap A = \emptyset$ , both actions with controllable and known durations;  $I$  is a function  $I : P \cup V \rightarrow \{\top, \perp\} \cup \mathbb{R}$  describing the initial state;  $G$  is a set of goals, where goals in  $G$  are Boolean variables from  $P$  or inequalities over real variables in  $V$  that represent the objectives that must be achieved;  $T$  is a set of TILs.

The TCP problem includes two type of actions, where physical actions can be defined as durative actions (see Definition 1), while durative sensing actions in  $\Delta$  are an special type of sensing actions (see Definition 3) defined as follows:

**Definition 7.** A durative sensing action  $\delta$ , where  $\delta \in \Delta$ , is a durative action with a nondeterministic outcome defined by the tuple  $\langle \delta_{pre}, \delta_{eff}, \delta_{dur} \rangle$ .  $\delta_{pre}$  is a set of conditions, including a set of literals, that must hold for the action to be applicable of type: at-start, over-all, and at-end.  $\delta_{eff}$  includes a set of possible action outcomes  $\delta'_{eff}$  and  $\delta''_{eff}$  that include effects of positive, negative, and numeric startings, continuous numeric, positive, negative, and numeric endings. The possible action outcomes characterise an unknown literal  $l$ , by considering the possible outcomes for  $l$  ( $l$  or  $\neg l$ ) described in a belief state  $S$ .

In a belief state  $S_A = \{(l, l_1, l_2), (\neg l, l_1, l_2), (l, \neg l_1, l_2), (\neg l, \neg l_1, l_2)\}$ , there would be states where  $l$  is possible and states where  $\neg l$  is possible so  $l$ 's true value is unknown.  $\delta$ 's outcomes  $\delta'_{eff}$  and  $\delta''_{eff}$  define two independent belief states  $S'_B = \{(l, l_1, l_2), (l, \neg l_1, l_2)\}$  and  $S''_B = \{(\neg l, l_1, l_2), (\neg l, \neg l_1, l_2)\}$ .  $\delta_{dur}$  represents a set of duration constraints (controllable and known). A sensing action  $\delta$ , where  $\delta \in \Delta$ , helps to completely characterise an unknown literal  $l$ , where  $l \in I$ , by considering the possible outcomes for  $l$  described in a belief state  $S'$ . The solution of the TCP problem is a transition tree which is defined based on the outcomes of the sensing action.

**PDDL Encoding.** We propose a set of extensions to PDDL2.1, which define the unknown literals, the set of possible outcomes of a durative sensing action, and the connection between the literal sensed and the effects of the sensing action. This approach is inspired by previous approaches that represent incomplete information, and sensing [29, 39]. Our PDDL extensions to encode the TCP problem are: (i) **:unknown-literals.** Figure 2 (left top) shows an example of how incomplete information is represented for Example 1. In `:unknown-literals`, the incomplete information in the problem is identified, which can be extracted for the initial state  $I$ . For instance, the domain the unknown literals reflect that the valve states are initially unknown, `(state_on ?v - valve)`; (ii) **:knowledge-updates.** Figure 2 (left bottom) shows an example of the observational effects extension for the Example 1. In `:knowledge-updates`, the `and` and `oneof` clauses [23] are used to define the set of facts corresponding to the possible outcomes of a sensing action. For the Example 1 domain, the unknown literal `(state_on v1)` is associated with two possible effects, first, the valve's state is open `((state_on v1))`, the second, the valve's state is closed `((not (state_on v1)))`. Notice that in the second effect, the `and` clause is used to specify the state of two literals, `(not (state_on v1))` and `(valve_closed wp32)`; and (iii) **:durative-action.** Finally, we present the extension for sensing outcomes in Figure 2 (right). In `:durative-action`, a durative sensing

<pre> (<b>:unknown-literals</b>  (state_on v1)  (state_on v2)  )  (<b>:knowledge-updates</b>  (oneof (state_on v1)         (and (not (state_on v1))               (valve_closed wp32)))  (oneof (state_on v2)         (and (not (state_on v2))               (valve_closed wp42)))  ) </pre>	<pre> (<b>:durative-action sense-valve</b>  :parameters (?r - robot ?s - sensor               ?v - poi ?wp - waypoint)  :duration ( = ?duration 5)  :condition (and               (over all (can_sense ?r ?s))               (...))  :effect (and            (at end (available ?r))            (...))  <b>:observe (and (at end (state_on ?v)))</b>  ) </pre>
--	--

Figure 2: PDDL extensions for representing incomplete information and observational effects in durative actions using constructs `:unknown-literals` (left top) and `:knowledge-updates` (left bottom).

action is extended from an ordinary (physical) action to include the `:observe` construct, which connects the action to the sensed literal. In the example, the sensing action (`sense-valve`) observes the valve's state (`state_on ?v - valve`) for a particular valve `?v` which gives rise to multiple possible outcomes, such as those in Figure 1 (right). This action includes time constraints and is not limited to Boolean effects on literals. A durative sensing action in a TCP problem is nondeterministic based on the sensing outcomes from the `:observe`. However, a sensing action contains another set of deterministic effects that are associated with the `:effect`. For instance, the positive ending effect (`at end (available ?r)`), which defines the robot is available when the action finishes. Considering this, the set of belief states associated with the outcome of a sensing action contains an effect associated with the unknown literal (different for each outcome) and a set of effects that are the same for all possible belief states. In Example 1, the possible outcomes from implementing the sensing action in valve `v1` are: (i) `(and (state_on v1) (available auv1) (v_at v1 wp32) ...)`, and (ii) `(and (not (state_on v1)) (valve_closed wp32) (available auv1) (v_at v1 wp32) ...)`.

## 4 TraCE Planning

The TraCE planner takes as input a simple PPOS problem with temporal and numeric constraints  $P_{TC}$  and generates a time-knowledge-aware plan  $\Pi_{TC}$  defined as follows:

**Definition 8.** A time-knowledge-aware plan  $\Pi_{TC} = (N, E)$  for a temporally-contingent planning problem  $P_{TC}$  is a transition tree  $B$ , represented as an AND/OR graph, where nodes  $N$  are labelled with actions built on a set of tuples,  $\pi_A := \langle a, t, d \rangle$  for physical actions and  $\pi_\Delta := \langle \delta, t, d \rangle$  for sensing actions; and edges  $E$  represent the action outcomes, denoting the set of propositions whose value are known after an action execution, where  $a \in A$  is an instantaneous or durative action,  $\delta \in \Delta$  is a durative sensing action,  $t$  is the action starting time,  $d$  represents the action duration,  $t \in \mathbb{R}_{\geq 0}$ , and  $d \in \mathbb{R}_{> 0}$  when actions have a duration.

The plan  $\Pi_{TC}$  arising from Definition 8 describes a tree where physical and sensing actions are encoded as two different types of vertices: (i)  $v_A$ , denoting vertices that describe physical actions, where  $v(a)$  is a vertex,  $v(a) \in v_A$ , and  $a$  is a physical action,  $a \in A$ ; and (ii)  $v_\Delta$ , which describes sensing action vertices, where  $v(\delta)$  is a vertex,  $v(\delta) \in v_\Delta$ , and  $\delta$  is a sensing action,  $\delta \in \Delta$ . The  $B$  tree edges capture the action ordering, where an edge  $z(v(x), v(y))$  defines that the action denoted by the vertex  $v(x)$  is executed

**Algorithm 1:** TRACE PLANNER ( $D_{TC}, P_{TC}, U$ )

---

**Output:**  $\Pi_{TC}$  (Branched Temporal Plan)

```

1 begin
2    $\Pi_T \leftarrow \emptyset$ 
3    $\Pi_T^*, H \leftarrow \text{FINDTEMPORALPLAN}(D_{TC}, P_{TC}, U, t_{\text{initial}})$ 
4   if  $\text{CheckExistence}(\Pi_T^*)$  then
5     while  $\Pi_T \neq \Pi_T^*$  do
6        $\Pi_T, H \leftarrow \text{Update}(\Pi_T^*, H)$ 
7        $b, Q \leftarrow \text{BUILDBRANCH}(D_{TC}, U, \Pi_T, H)$ 
8        $root \leftarrow \text{EXPANDTREE}(D_{TC}, P_{TC}, b, H, U, Q)$ 
9       return  $\Pi_{TC} \leftarrow root$ 
10  else
11    return FAILURE

```

---

before the action denoted by  $v(y)$ . Physical actions are deterministic; therefore a vertex  $v(a)$  has at most a single edge. Sensing actions are nondeterministic; therefore, a vertex  $v(\delta)$ , associated with the sensing action  $\delta$ , is characterised by at least two possible outgoing edges. In this case, the number of edges depends on the possible  $\delta$  outcomes.

Algorithm 1 shows the TraCE planning approach, which returns a time-knowledge-aware plan  $\Pi_{TC}$ . TraCE is a PDDL planner that aims to support the PDDL extensions modelling the TCP problem in full. The required inputs to the TraCE solver are the domain  $D_{TC}$ , including physical and sensing actions; the TCP problem  $P_{TC}$ , which defines the initial state; and  $U$ , which includes the incomplete information and observational effects.  $U$  incorporates the incomplete information in the `:unknown-literals`'s body and the observational effects in the `:knowledge-updates`'s body; while the sensing actions in  $D_{TC}$  include the extension for sensing outcomes. The branches in  $\Pi_{TC}$  are time-aware plan solutions ( $\Pi_T$ ) that solve a temporal planning problem  $P_T$  under different sets of contingent outcomes arising from domain incompleteness and durative sensing actions. For instance, the solution for Example 1 shows the branches of a  $\Pi_{TC}$ . Top and bottom branches represent two different time-aware plans considering the sensing action `sense-valve` outcomes define the valves (`v1` and `v2`) are (both) closed and open, respectively.

The strategy starts by initialising an empty temporal plan solution  $\Pi_T$  (line 2), which preserves the temporal plan associated with a particular branch. Then, the approach finds an ordinary (initial) deterministic temporal plan  $\Pi_T^*$  (line 3) for the problem that represents the largest temporal plan solution. In this work, the *plan size* is defined as the total number of actions in a plan (that describe each branch). Therefore, the largest plan is the solution with more actions. For instance, in Example 1, the largest deterministic temporal plan is represented by the bottom branch. `FINDTEMPORALPLAN` computes the largest plan and returns it in  $\Pi_T^*$ . In our approach, the initial temporal plan  $\Pi_T^*$  can be updated in parallel to the  $\Pi_{TC}$  search (line 4-9).

The history set  $H$  for the temporal plan denotes the state  $s[i]$  at which  $\Pi_T[i]$  or  $\Pi_T^*[i]$  is executed, and the time  $t[i]$  at which  $i$  starts and concludes, where  $H[i] = \langle s[i], t[i] \rangle$  and  $\Pi_T[i]$  or  $\Pi_T^*[i]$  defines the  $i$ th plan's action in  $\Pi_T$  or  $\Pi_T^*$ .  $H[i+1]$  describes the possible states reached after an execution of  $\Pi_T[i]$  or  $\Pi_T^*[i]$ . Then, if  $\Pi_T^*$  exists (line 4), TraCE checks if  $\Pi_T \neq \Pi_T^*$  (line 5). Notice that the condition  $\Pi_T \neq \Pi_T^*$  allows the strategy to update the initial temporally-contingent plan solution if after generating a plan  $\Pi_{TC}$ ,  $\Pi_T^*$  differs from the first  $\Pi_T$  as a result  $\Pi_T^*$  was optimised. If the strategy finds discrepancies between  $\Pi_T$  and  $\Pi_T^*$ , the first is updated with the plan in  $\Pi_T^*$  (line 6). Then, the strategy builds a branch  $b$  using the `BUILDBRANCH` method (see Algorithm 2) and expands the tree for a set of "tasks" describing



the sensing actions in  $b$  to obtain  $root$  (line 7-8) using the EXPANDTREE method (see Algorithm 3). The variable  $root$  (identified by first action in the plan) describes  $\Pi_{TC}$ , and  $Q$  defines a queue of tasks  $\langle n, u \rangle$ , where  $n \in N$  is a sensing node (representing a sensing action) and  $u$  is a possible outcome of the sensing action, where  $u \in U$ . Each  $q \in Q$  introduces different subsets of initial conditions based on the sensing action outcomes that leads to a new temporal planning problem  $P_T$ . After the tree expansion, the  $\Pi_{TC}$  is returned (line 9). If  $\Pi_T^*$  is not found the planner fails to find a solution for  $P_{TC}$  (line 10-11).

The FINDTEMPORALPLAN method computes temporal plans using the OPTIC [2] planner. TraCE branch generation considers temporal-numeric planning within a forward state space search framework. FINDTEMPORALPLAN does not consider changes in the temporal planning search regarding the OPTIC approach. Instead, the strategy provides the right conditional and time constraint inputs to the propositional planning techniques and the sub-solver used to schedule the action sequence to generate the required time-aware plan. The algorithm finds the plan with the largest size by iterating over the initial state, considering the combinations of incomplete information and observational effects. We solve the plan length check by including into the problem initial state the set of observational effects that defines the unknown literals in  $U$  to be false ( $S_o^-$ ). Considering this information in the initial state we obtain a plan solution  $\Pi_T^-$ . Then, the plan  $\Pi_T^-$  is stored (if it exists). A similar procedure is repeated considering all sensing outcomes associated with the unknown literals are true ( $S_o^+$ ) and we achieve a plan solution  $\Pi_T^+$  for this initial state. The approach compares the two plan solutions ( $\Pi_T^-$  and  $\Pi_T^+$ ) and it saves the largest plan in  $\Pi_T^*$ . The planning problems that we consider include temporal and numeric constraints that force actions in some order in the plan solution. This justifies our decision to start the branch generation using the largest plan as this solution satisfies the problem with the most considerable set of temporal and numeric restrictions. Temporal planners attempt to improve the initial plan solution, usually considering as a metric the makespan minimisation.

The FINDTEMPORALPLAN method returns the plan's history  $H$ . The second component of the  $H$  is the time  $t[i]$  associated with the action execution. The core of the temporal planning solver implements the scheduling of the plan's actions considering STNs and LP methods. We refer to these strategies as a temporal solver (TS). The TS represents the temporal constraints between the set of time points where actions start or end. Therefore, we can describe the occurrence of an action  $i$  in the TS reasoning as the time point when the action starts  $i_+$  and  $i_-$  when the action ends. This information is saved in  $t[i] = [i_+, i_-]$ . The FINDTEMPORALPLAN strategy extends the scheduling analysis by controlling the  $t_{initial}$  value. The  $t_{initial}$  is introduced as an input to the approach (see line 3 in Algorithm 1). For the first temporal plan  $\Pi_T^*$ ,  $t_{initial}$  is considered 0. However, this modification gains importance when expanding the tree based on the sensing vertices. In these cases, the time the sensing action concludes is defined as the  $t_{initial}$  for a new branch in the tree associated with a different outgoing edge from the sensing vertex. We put this into context when describing the remaining parts of the algorithm. The TS encodes a TIL by adding a time point  $t(p)$  for the occurrence of TIL  $p$ , with the temporal constraint  $t(p) - t_{initial} = time_{TIL}(p)$ , where  $time_{TIL}(p)$  is the time at which  $p$  occurs. The TILs describe other time points that need to be ordered by the TS approach when scheduling the plan's actions.

The BUILDBRANCH method in Algorithm 2 describes the branch generation, which starts with the first action in  $\Pi_T$ . The strategy initialises  $Q$  (line 2) and creates the first branch over the  $\Pi_T$ 's size (line 3). The branch creation relies upon the definition of all nodes in  $\Pi_T$  (vertices) and their connections (edges). For each action  $\Pi_T[i]$  in the plan the algorithm initialises a node  $n$  (line 4) that includes a set of *definition parameters*: (i)  **$n.name$** . is the name of the action describing the node; (ii)  **$n.s-action$** . Boolean variable to define if the node represents a sensing action; (iii)  **$n.parent$** . is the node  $n$  parent; and  **$n.depth$** . number of edges from the root node to  $n.name$ . The strategy takes the information that the FINDTEMPORALPLAN method saves in the history  $H$  (state and time) to define another two node's

parameters (line 5): (iv)  $n.state$ . defines the state at which the action in  $n.name$  is executed; and (v)  $n.time$ . defines the time at which the node (action) starts and concludes. If the plan size is 0 (line 6-7), the branch  $b$  contains precisely the node  $n$ , which contains the set of parameters initialised. On the contrary, the TraCE creates an outgoing edge from node  $n$  to another new node  $child$  that denotes the next action (line 8-9). Then, the method defines the state and time<sup>1</sup> parameters for the child (line 10). TraCE appends the node  $n$  children in the node parameter (line 11).  $n.children$  defines the set of  $n.name(s)$  (actions) associated with a  $n.name$ . If the action to label is a sensing action (line 12), TraCE maps the edge from node  $n$  to child  $child$  with the relevant outcome of the sensing action, obtained from the history  $H$  of  $\Pi_T$  and  $U$  (line 13-14). This labelling is stored in the last node  $n$  definition parameter:  $n.edge-map$ . is the label of the outgoing edge from the node to its child, where  $n$  is a sensing node.

The node  $n$ , describing a sensing action in  $\Delta$ , and the possible outcomes defined in  $U$  for the action that does not label any outgoing edge from  $n$ , describes a new temporally-contingent planning task with an initial state  $I^i$  obtained from  $H[i]$ . The planner stores the sensing nodes and their possible outcomes ( $\langle n, u \rangle$ ) obtained when building the branch  $b$  in  $Q$  (line 15-16) The algorithm returns  $b$  and an updated  $Q$  (line 17). The  $n.depth$  definition parameter helps to organise the tasks in the queue  $Q$  to implement the expansion.

---

**Algorithm 2: BUILD BRANCH**


---

**Input:**  $\langle D_{TC}, U, \Pi_T, H \rangle$   
**Output:**  $\langle b, Q \rangle$

```

1 begin
2    $Q \leftarrow \emptyset$ 
3   for  $i = 0, \dots, size(\Pi_T) - 1$  do
4      $n \leftarrow CreateNode(\Pi_T[i])$ 
5      $n.state, n.time = H[i]$ 
6     if  $i == 0$  then
7        $b == n$ 
8     if  $i \neq size(\Pi_T) - 1$  then
9        $child \leftarrow CreateNode(\Pi_T[i + 1])$ 
10       $child.state, child.time = H[i + 1]$ 
11       $n.children \leftarrow AppendChildToNode(child)$ 
12      if  $n.s-action$  then
13         $U_\Delta \leftarrow ObtainCurrentOutcome(H[i] \cap U_{\Pi_T[i]})$ 
14         $n.edge-map(child) = u$ 
15        for  $u = U_\Delta \setminus \{u_\Delta\}$  do
16           $Q \leftarrow AddToQueue(Q, \langle n, u \rangle)$ 
17  return  $b, Q$ 

```

---



---

**Algorithm 3: EXPAND TREE**


---

**Input:**  $\langle D_{TC}, P_{TC}, b, H, U, Q \rangle$   
**Output:**  $root$

```

1 begin
2   while not  $Q \leftarrow \emptyset$  do
3     for  $\langle n_i, u_i \rangle \leftarrow ExtractFromQueue(Q)$  do
4        $I = n_i.state$ 
5        $I'' \leftarrow ModifyInitialState(I, u_i)$ 
6        $P_{TC}'' \leftarrow update.P_{TC}(I'')$ 
7        $\Pi_T, H \leftarrow FINDTEMPORALPLAN(P_{TC}'', U, n_i.time)$ 
8       if  $CheckExistence(\Pi_T)$  then
9          $branch\_child, Q'' \leftarrow BUILD\_BRANCH(D_{TC}, U, \Pi_T, H)$ 
10         $branch\_child.edge-map = u_i$ 
11         $n_i.children \leftarrow AddToTree(branch\_child)$ 
12         $Q \leftarrow update.Q(Q'')$ 
13      else
14         $root \leftarrow \emptyset$ 
15        return BREAK
16  return  $root$ 

```

---

The EXPANDTREE method in Algorithm 3 evaluates the task set  $Q$  sequentially considering the order imposed by the depth to compute the tree expansion (line 2-3). For each  $q \in Q$ , the approach defines the initial state  $I$  using the node information (line 4) and modifies  $I$  according to the outcome  $u_i$  (line 5).  $P_{TC}$  is updated with  $I''$  (line 6). Then, TraCE triggers the temporal planning search (line 7), which finds the largest plan solution considering the current initial state  $I''$ . Here, the branch is not the first; therefore  $t_{start}$  is not 0. The FINDTEMPORALPLAN method finds sequences of actions that satisfy the propositional

<sup>1</sup>At this stage  $t[i + 1]$  for an action  $i$  includes initial time of the next action which is  $t_i + \epsilon$ .

preconditions considering the (new) initial state information. Then, the planning approach uses the TS method to schedule the action sequence considering the starting time for the new plan is  $t_{start} = n_i$ .time.  $H$  saves the right time information regarding each action in the temporal plan  $\Pi_T$ . Supposing  $\Pi_T$  exists (line 8), the method creates a new branch *branch\_child* of the tree (new child) and finds the task queue  $Q$  associated with the branch (line 9) using Algorithm 2. The branch is mapped (line 10) by labelling the outgoing edges from the node to its child, considering  $u_i$  information. The node children are then added to the tree (line 11). The strategy updates the task queue  $Q$  to compute further branches considering the new sensing nodes identified in the *branch\_child* (line 12). If the approach does not find a plan the EXPANDTREE returns an empty plan in *root* (line 13-15).

We implement the tree expansion sequentially and all branches expanded must return a solvable temporal plan. The algorithm returns a *root* (line 16) when all branches are fully expanded. This occurs when the  $Q$  set is empty and all possible branches are created. The *root* builds over the set of nodes explored during the tree generation. The node labelling and depth information are fundamental to obtaining the transition tree. Saving the times at which sensing actions are executed allows the strategy to maintain the action scheduling in all branches. The TraCE planner iterates the solution by considering possible updates in the initial deterministic plan solution. If the algorithm identifies an update in the  $\Pi_T^*$  at the time the  $\Pi_{TC}$  is achieved, the planner starts the process again. In this case, TraCE starts finding a new *root* based on a new initial temporal plan solution. Notice that if an updated  $\Pi_T^*$  exist, the process starts again independently on the plan in *root*. Therefore, if the last *root* was empty, it does not affect the search for a new plan. For one planning iteration, if the computed plan presents a branching factor  $\mu$  and its depth (maximum) is  $\eta$ , the tree describing  $\Pi_{TC}$  has at most  $\mu^\eta$  leaves, where the leaves are physical vertices. Therefore, the TraCE planner calls the temporal planning algorithm at most  $\mu^\eta$  when finding a solution. TraCE provides a generic solution. The planner solves problems with temporal, numeric and perception requirements. However, it can also solve problems with fully-known initial states as a common temporal planning problem. Soundness and completeness results for our approach are presented in Appendix A.

## 5 Experimental Evaluation

Our domain and problems are encoded in PDDL. All experiments in this section are run on Ubuntu 16.04, with an Intel Core i7-8700, limiting the planner to 30 minutes of CPU@3.2GHz, 16GB of RAM. We illustrate the TraCE planner performance in three experiments that cover offline planning and execution: **Experiment 1.** This experiment compares the planner with a state-of-art offline conditional planner PO-RPR, to evaluate the efficiency of the proposed approach generating solutions for a set of simple PPOS domains<sup>2</sup>, including Coloured Balls (cballs), Canadian Traveller’s Problem (ctp-ch), and Doors (doors). These domains are described in [34]. TraCE uses a revised version of the domains considering durative actions; numeric constraints are not included.

**Real-World Domains.** Experiment 2-3 considers planning domains motivated by real-world domains<sup>3</sup>, particularly robotics applications, including the Offshore Energy Platform (oep), Manufacturing Plant (mp), Valve Manipulation (vm) and the Neighbourhoods (n) domains. All these domains describe robotics problems where temporal requirements (e.g., TILs and deadlines), numeric constraints, and partial observability are part of the mission characteristics.

**Experiment 2.** Here, we evaluate the quality of the planning algorithm while generating plans for real-world domains. The problems introduce a set of robotic scenarios where robots need to implement tasks

<sup>2</sup>The original problems and PO-PRP source are presented in <https://github.com/QuMuLab/planner-for-relevant-policies.git>

<sup>3</sup>These domains and problems can be found at <https://github.com/YanielCarreno/tcp-domains>.

Problem	Size ( $\Delta + A$ )		PT	
	PO-PRP	TraCE	PO-PRP	TraCE
cballs-4-1	261	232	0.02	3.45
cballs-4-2	13887	TO	0.67	TO
cballs-10-1	4170	TO	1.57	TO
ctp-ch-1	4	6	0.00	2.01
ctp-ch-5	16	16	0.00	7.05
ctp-ch-10	31	35	0.02	7.10
ctp-ch-15	46	42	0.07	8.24
doors-5	82	96	0.01	3.00
doors-7	1295	2291	0.04	16.90
doors-9	28442	TO	1.07	TO

Table 1: Experiment 1: Plan size and planning time (PT) (sec) for PO-PRP and TraCE when solving 10 simple PPOS problem instances. TO indicates time out.

Problem	oep			mp			vm			bc		
	BF	Size	PT	BF	Size	PT	BF	Size	PT	BF	Size	PT
1	4	85	41.50	2	78	32.30	2	44	13.40	2	42	9.05
2	3	77	22.15	3	57	47.50	2	64	28.12	2	156	28.63
3	2	66	27.23	2	108	32.52	2	67	21.23	2	108	33.42
4	2	81	32.03	4	72	42.29	2	95	19.92	2	88	14.81
5	2	102	35.50	3	340	69.43	2	145	12.24	2	234	44.70
6	3	90	39.34	4	75	52.18	2	132	14.14	2	69	8.44
7	3	123	53.20	3	72	50.90	2	80	17.15	2	62	6.13
8	4	116	66.23	3	110	84.32	2	109	27.08	2	92	19.13
9	5	TO	TO	4	145	54.91	2	56	18.83	2	134	21.12
10	6	TO	TO	5	TO	TO	2	134	96.90	2	287	48.32

Table 2: Experiment 2: Branching factor (BF), plan size (Size), and planning time (sec) (PT), including physical and sensing actions, for 10 problem instances of the real-world robotic application domains using TraCE. TO indicates time out.

with temporal and numeric constraints to maintain the operation, and sensing actions are required to solve the planing problem.

**Experiment 3.** We evaluate the system’s performance in a laboratory environment using a BlueROV2. The experiment analyses the mission execution time associated with plans that include non-deterministic sensing actions. Here the problem goals include the manipulation of five valves, sensors need to check the valves’ state before executing the manipulation. We compare TraCE and OPTIC plan execution. The execution framework embedding both planners support replanning during the mission. The experiment evaluates ten different problems where we introduce five forced failures during the mission, associated with the valve state and valve localisation.

## 6 Discussion

The results for Experiment 1 are presented in Table 1. The TraCE planner finds a solution for most problem instances solved by PO-PRP in the experimental domains. The planner obtains plans with sizes similar to PO-PRP’s plan sizes in several problems (highlighted in red). However, PO-PRP outperforms TraCE in terms of planning times. TraCE sequentially expands the tree, introducing a delay in plan generation. In addition, the TraCE planner uses temporal solvers to create the branches and expand the tree, which affects the planning times. Temporal planners must meet the causality and temporality constraints imposed by the model to obtain a valid plan. Therefore, additional time delays using TraCE compared to PO-PRP might be expected, as the PO-PRP planner does not consider the time to schedule the action sequence when finding a plan solution. We introduce time reasoning to address another set of problems that PO-PRP cannot solve. Here, we present the results for the first  $\Pi_{TC}$  obtained. Additional iterations optimise the plan solution by reducing the plan size; however, this introduces additional delays in the plan generation.

Table 2 shows that TraCE solves most of the problem instances. The planner generates solutions with different branching factors, demonstrating that TraCE deals with complex problems and scales well. As expected, we note that the size of the plans and the planning times rise with the increase in the number of sensing actions required in the plan to reduce the knowledge incompleteness. The oep and mp domain

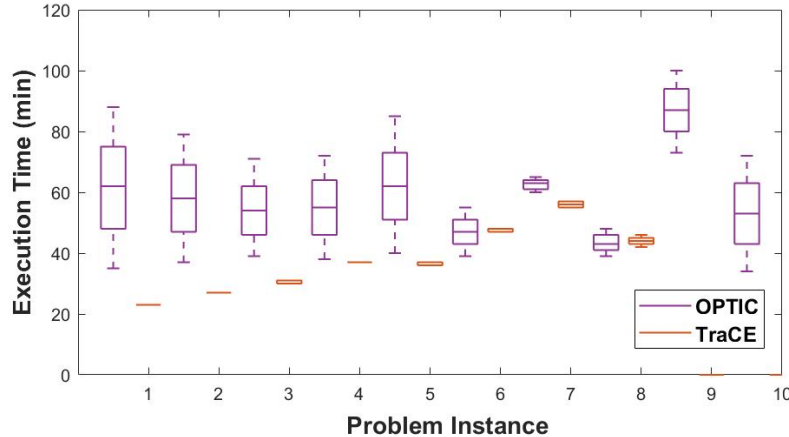


Figure 3: Experiment 3: Plan execution time (min) in 10 problem instances of the Example 1 domain over 10 runs using TraCE and OPTIC.

incompleteness leads to more significant branching factors and planning times as the robots can explore multiple paths (for the oep domain) and control several flows in a valve (for the mp domain). TraCE planning times for the problem instances of the oep and mp domains are significantly higher compared to the other two domains. These domains require the robots to find plans that deal with concurrent action execution and numeric constraints along the branches which also affects planning times. The planner finds solutions for all problem instances of the vm and bc domains. Although those domains present small branching factors, planning times increase for complex problems (see problems 9 and 10). The main reason for this increase is that TraCE needs to find a solution for problems where the robot needs to recharge to maintain operation.

Figure 3 shows that the generation of contingent branches during the planning stage significantly improves mission implementation times. This is mainly because OPTIC needs to replan every time a discrepancy between the real value from the sensor (run-time) and the expected outcome (planning time) is found. Replanning introduces unnecessary delays in the mission, mainly when we can use alternative algorithms that deal with certain levels of uncertainty in the domain. Our approach reduces replanning in non-quiescent environments since TraCE introduces the contingency analysis that specifies the possible outcomes from the observation.  $\Pi_{TC}$  can be affected by noisy sensors, which might force the system to generate a new plan if the sensing action does not provide a conclusive outcome.

## 7 Conclusions

We present the TraCE planner, which deals with incomplete knowledge, sensing, temporal notions, and numeric constraints. Our approach combines temporal plan construction in a contingent planning framework, offering more robust solutions for new types of applications. We can model problems that (i) require temporal reasoning, such as timed initial literals and deadlines; (ii) manage resources, using numerical fluents, offering more powerful modelling of mission scenarios; and (iii) include partial observability. We propose a PDDL extension to model sensing actions that operate over unknown propositions, and possible outcomes from non-deterministic sensing actions. We evaluate the approach in experiments that cover offline planning and execution in a new set of domains.

## References

- [1] Alexandre Albore, Hector Palacios & Hector Geffner (2009): *A translation-based approach to contingent planning*. In: *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, AAAI Press, pp. 1623–1628.
- [2] J Benton, Amanda Jane Coles & Andrew Coles (2012): *Temporal Planning with Preferences and Time-Dependent Continuous Costs*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 2–10.
- [3] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri & Paolo Traverso (2001): *Planning in nondeterministic domains under partial observability via symbolic model checking*. In: *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001, AAAI Press, pp. 473–478.
- [4] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri & Paolo Traverso (2006): *Strong planning under partial observability*. *Artificial Intelligence* 170(4-5), pp. 337–384, doi:10.1016/j.artint.2006.01.004.
- [5] Blai Bonet & Hector Geffner (2011): *Planning under partial observability by classical replanning: Theory and experiments*. In: *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, AAAI Press.
- [6] Daniel Bryce, Subbarao Kambhampati & David E Smith (2006): *Planning graph heuristics for belief space search*. *Journal of Artificial Intelligence Research (JAIR)* 26, pp. 35–99, doi:10.1613/jair.1869.
- [7] Alberto Camacho, Christian J Muise & Sheila A McIlraith (2016): *From FOND to Robust Probabilistic Planning: Computing Compact Policies that Bypass Avoidable Deadends*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 65–69.
- [8] Y. Carreno, A. Ng Jun Hao, Y. Petillot & R. P. A Petrick (2022): *Planning, Execution, and Adaptation for Multi-Robot Systems using Probabilistic and Temporal Planning*. In: *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, IFAAMAS, pp. 217–225.
- [9] Y. Carreno, È. Pairet, Y. Petillot & R. P. A Petrick (2020): *Task allocation strategy for heterogeneous robot teams in offshore missions*. In: *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, IFAAMAS, pp. 222–230.
- [10] Y. Carreno, Y. Petillot & R. P. A. Petrick (2020): *Towards Robust Mission Execution via Temporal and Contingent Planning*. In: *Proc. of the Annual Conference Towards Autonomous Robotic Systems (TAROS)*, Springer, pp. 214–217, doi:10.1007/978-3-030-63486-5\_24.
- [11] Y. Carreno, Y. Petillot & R. P. A Petrick (2021): *Compiling Contingent Planning into Temporal Planning for Robust AUV Deployments*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Planning and Robotics (PlanRob)*.
- [12] Y. Carreno, J. Scharff Willners, Y. Petillot & R. P. A Petrick (2021): *Situation-Aware Task Planning for Robust AUV Exploration in Extreme Environments*. In: *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Robust and Reliable Autonomy in the Wild (R2AW)*.
- [13] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni & Bram Ridder (2017): *Opportunistic planning in autonomous underwater missions*. *IEEE Transactions on Automation Science and Engineering* 15(2), pp. 519–530, doi:10.1109/TASE.2016.2636662.
- [14] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli & Marco Roveri (2014): *Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty*. In: *Proc. of the AAAI Conference on Artificial Intelligence*, 28.
- [15] Amanda Jane Coles (2012): *Opportunistic Branched Plans to Maximise Utility in the Presence of Resource Uncertainty*. In: *Proc. of the European Conference on Artificial Intelligence (ECAI)*, 2012, p. 252.
- [16] Amanda Jane Coles, Andrew Coles, Maria Fox & Derek Long (2010): *Forward-Chaining Partial-Order Planning*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 42–49.

- [17] Carlo Combi, Luke Hunsberger & Roberto Posenato (2013): *An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty*. *Evaluation* 1(1).
- [18] Carlo Combi, Roberto Posenato, Luca Viganò & Matteo Zavatteri (2019): *Conditional simple temporal networks with uncertainty and resources*. *Journal of Artificial Intelligence Research (JAIR)* 64, pp. 931–985.
- [19] Stephen Cresswell & Alexandra Coddington (2003): *Planning with timed literals and deadlines*. In: *Proc. of Workshop of the UK PlanSIG*, pp. 23–35.
- [20] Patrick Eyerich, Robert Mattmüller & Gabriele Röger (2009): *Using the context-enhanced additive heuristic for temporal and numeric planning*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 130–137.
- [21] Janae N Foss & Nilufer Onder (2005): *Generating temporally contingent plans*. In: *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains*.
- [22] Maria Fox & Derek Long (2003): *PDDL2.1: An extension to PDDL for expressing temporal planning domains*. *Journal of Artificial Intelligence Research (JAIR)* 20, pp. 61–124, doi:10.1613/jair.1129.
- [23] Hector Geffner & Blai Bonet (2013): *A concise introduction to models and methods for automated planning*. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(1), pp. 1–141, doi:10.2200/S00513ED1V01Y201306AIM022.
- [24] Malik Ghallab, Dana Nau & Paolo Traverso (2004): *Automated Planning: theory and practice*. Elsevier.
- [25] Muhammad Adnan Hashmi & Amal El Fallah Seghrouchni (2010): *Merging of temporal plans supported by plan repairing*. In: *Proc. of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2, IEEE, pp. 87–94.
- [26] Malte Helmert (2009): *Concise finite-domain representations for PDDL planning tasks*. *Artificial Intelligence* 173(5-6), pp. 503–535, doi:10.1016/j.artint.2008.10.013.
- [27] Karla L Hoffman, Manfred Padberg & Giovanni Rinaldi (2013): *Traveling salesman problem*. In: *Encyclopedia of operations research and management science*, Springer, pp. 1573–1578, doi:10.1007/978-1-4419-1153-7\_1068.
- [28] Jörg Hoffmann (2003): *The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Variables*. *Journal of Artificial Intelligence Research (JAIR)* 20, pp. 291–341, doi:10.1613/jair.1144.
- [29] Jörg Hoffmann & Ronen Brafman (2005): *Contingent planning via heuristic forward search with implicit belief states*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 71–80.
- [30] L. Kunze, N. Hawes, T. Duckett, M. Hanheide & T. Krajník (2018): *Artificial intelligence for long-term robot autonomy: A survey*. *IEEE Robotics and Automation Letters (RA-L)* 3(4), pp. 4023–4030, doi:10.1109/LRA.2018.2860628.
- [31] Derek Long & Maria Fox (2003): *The 3rd international planning competition: Results and analysis*. *Journal of Artificial Intelligence Research (JAIR)* 20, pp. 1–59, doi:10.1613/jair.1240.
- [32] Derek Long & Maria Fox (2003): *Exploiting a graphplan framework in temporal planning*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 51–62.
- [33] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld & David Wilkins (1998): *PDDL – The Planning Domain Definition Language (Version 1.2)*. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.
- [34] Christian Muise, Vaishak Belle & Sheila McIlraith (2014): *Computing contingent plans via fully observable non-deterministic planning*. In: *Proc. of the AAAI Conference on Artificial Intelligence*, 28.
- [35] Christian Muise, Sheila McIlraith & Christopher Beck (2012): *Improved non-deterministic planning by exploiting state relevance*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 172–180.

- [36] Ahmed Nouman, Volkan Patoglu & Esra Erdem (2021): *Hybrid conditional planning for robotic applications*. *The International Journal of Robotics Research* 40(2-3), pp. 594–623, doi:10.1177/0278364920963783.
- [37] Héctor Palacios, Alexandre Albore & Hector Geffner (2014): *Compiling contingent planning into classical planning: New translations and results*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Models and Paradigms for Planning under Uncertainty*, AAAI Press.
- [38] Mark A Peot & David E Smith (1992): *Conditional nonlinear planning*. In: *AIPS*, Elsevier, pp. 189–197.
- [39] Ronald P. A. Petrick & Fahiem Bacchus (2002): *A Knowledge-Based Approach to Planning with Incomplete Information and Sensing*. In: *Proc. of the International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 212–222.
- [40] Ronald P. A. Petrick & Fahiem Bacchus (2004): *Extending the Knowledge-Based Approach to Planning with Incomplete Information and Sensing*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 2–11.
- [41] Son Thanh To, Tran Cao Son & Enrico Pontelli (2011): *Contingent planning as AND/OR forward search with disjunctive representation*. In: *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI Press, pp. 258–265.
- [42] Ioannis Tsamardinos, Thierry Vidal & Martha E Pollack (2003): *CTP: A new constraint-based formalism for conditional, temporal planning*. *Constraints* 8(4), pp. 365–388, doi:10.1023/A:1025894003623.
- [43] Daniel S Weld, Corin R Anderson & David E Smith (1998): *Extending graphplan to handle uncertainty & sensing actions*. In: *Proc. of the AAAI Conference on Artificial Intelligence*, pp. 897–904.
- [44] Ibrahim Faruk Yalciner, Ahmed Nouman, Volkan Patoglu & Esra Erdem (2017): *Hybrid conditional planning using answer set programming*. *Theory and Practice of Logic Programming* 17(5-6), pp. 1027–1047, doi:10.1017/S1471068417000321.
- [45] Matteo Zavatteri & Luca Viganò (2019): *Conditional simple temporal networks with uncertainty and decisions*. *Theoretical Computer Science* 797, pp. 77–101, doi:10.1016/j.tcs.2018.09.023.



## A Soundness and Completeness

Temporal planning requires meeting the causality and scheduling constraints imposed by a temporal planning problem while looking for a sequence of actions that reaches the goal state. To achieve a valid plan, temporal planners must consider the action duration constraints along an unbounded length timeline. The TraCE planner includes the temporal planning algorithm `FINDTEMPORALPLAN`, which inherits the soundness and completeness of `OPTIC` and its predecessor `POPF`. The TraCE planner relies on constructing a transition tree where individual branches denote deterministic temporal plan solutions. This solution involves solving temporal planning problems with different initial states and merging them into a tree. Possible outcomes of nondeterministic sensing actions describe the initial states. Then, for a temporal plan solution:

**Definition 9.** *A temporal plan is sound (with respect to a given domain and problem) if action preconditions are satisfied in their respective states, temporal constraints are met, and the final action produces a state where the goals are satisfied.*

**Definition 10.** *A temporal planner (or temporal planning algorithm) is defined as sound if every temporal plan that the planner generates (with respect to a given domain and problem) is a sound plan.*

**Definition 11.** *A planner is complete if for every planning domain/problem that has a solution, the planner is guaranteed to produce a plan.*

The `FINDTEMPORALPLAN` temporal planning approach reasons about the state representation. The state considers: (i) the ordered list of start events (actions that have started but not yet finished) and the collection of temporal constraints over the actions in the plan to reach the current world state. These two state description components are defined over the sets  $P$  and  $V$  that hold in the world and support the planning approach's soundness.

We can explore the soundness and completeness of the TraCE planner considering it builds on the temporal plan solutions generated by the sound and complete method `FINDTEMPORALPLAN`. The temporal planning approach uses the domain actions to find a *valid plan*—a plan obtained by a sound a complete planner—that reaches the goal state. We assume the PDDL actions in the domain description do not present delayed effects. This means that action effects in the next state are defined in terms of the current state of the world. Following these points, the TraCE planner soundness is defined as follows:

**Lemma 1.** *(Soundness). TraCE is a sound planner if every branch of the tree from the root to a leaf constructed incrementally is a valid temporal plan of physical and sensing actions.*

**Proof.** Let  $\Pi'_T = \langle a_0, a_1, \dots, a_n \rangle$  be a temporal plan that solves the temporal planning problem  $P_T$ , where an action  $a_i$  is a physical or a sensing action ( $0 \leq i < n$ ) and the last action  $a_n$  in the plan is a physical action. This plan is valid considering Definition 10 and Definition 11. Let  $H' = \langle \langle S_0, t_0 \rangle, a_0, \langle S_1, t_1 \rangle, a_1, \dots, \langle S_n, t_n \rangle, a_n, \langle S_{n+1}, t_{n+1} \rangle \rangle$  describe the history of  $\Pi'_T$ , where every action  $a_i$  is executed at a belief state  $S_i$  at a time  $t_i$  and reaches a belief state  $S_{i+1}$  at a time  $t_{i+1}$  ( $0 \leq i < n$ ).  $S_{n+1}$  represents the last belief state that is the goal state considering the problem  $P_T$  definition.

For every sensing action  $a_j$  in  $\Pi'_T$  ( $0 \leq j < n$ ) with an outcome  $u_j$  observed at  $\langle S_{j+1}, t_{j+1} \rangle$ , TraCE constructs a task  $\langle n_j, u_j \rangle$  and introduces it into the queue of tasks  $Q$ . Let  $\Pi''_T = \langle a_j = a'_0, a'_1, \dots, a'_n \rangle$  a temporal plan with an history  $H'' = \langle \langle S_j, t_j \rangle = \langle S'_0, t'_0 \rangle, a'_0, \langle S'_1, t'_1 \rangle, a'_1, \dots, \langle S'_n, t'_n \rangle, a'_n, \langle S'_{n+1}, t'_{n+1} \rangle \rangle$  computed for the planning problem characterised by this task. This plan is valid considering Definition 10 and Definition 11. Then, considering we are solving temporal planning problems with actions that do not present delayed effects, the sequence of actions  $\langle a_0, a_1, \dots, a_j, a'_1, \dots, a'_n \rangle$  is a valid sequential plan computed for a temporal planning problem  $P_T$ , with history  $\langle \langle S_0, t_0 \rangle, a_0, \langle S_1, t_1 \rangle, a_1, \dots, \langle S_j, t_j \rangle, a_j, \langle S'_1, t'_1 \rangle,$

$a'_1, \dots, \langle S'_{n'}, t'_{n'} \rangle, a'_{n'}, \langle S'_{n'+1}, t'_{n'+1} \rangle \rangle$ . The first part of the plan,  $\langle a_0, a_1, \dots, a_j \rangle$ , does not prevent the last part of the plan,  $\langle a'_1, \dots, a'_{n'} \rangle$ , and vice versa, considering  $\Pi'_T$  and  $\Pi''_T$  are sound and complete their union is sound and complete.

We consider that all contingencies describing the incomplete knowledge in the initial state are specified in  $U$ . The TraCE planner uses this information to construct a tree with temporal branches sequentially.

**Lemma 2.** (Completeness). *TraCE is a complete temporally-contingent planner if all known contingencies are specified by its input  $U$ , and incrementally constructs a temporal plan  $P_{TC}$ , such that the plan size of each sub-branch  $b$  computed by one call of the temporal planning approach is smaller than the size of the first temporal plan obtained  $\Pi_T^*$ .*

**Proof.** We first show that TraCE is complete considering (i) all known contingencies are specified by  $U$ , and (ii) the planner incrementally constructs a temporally contingent plan such that the size of each sub-branch  $b$ , computed by one call of the temporal planning approach is smaller than the size of the first temporal plan obtained  $\Pi_T^*$ . If all contingencies for a sensing action lead to a branch in the tree, then the tree cannot be extended further. Otherwise, for some sensing action contingency, there is no branch in the tree. This means that there does not exist a temporal plan whose plan's size is less than or equal to  $\Pi_T^*$ . Considering every temporal plan computed by FINDTEMPORALPLAN when building the tree  $B$  is connected<sup>4</sup>, such a branch cannot be reconstructed. If there is not a  $\Pi_{TC}$  under the conditions we have mentioned in this proof, TraCE returns a failure. If there does not exist a temporally-contingent plan under the conditions and TraCE returns  $B$ , at least one branch of the tree (excluding  $\Pi_T^*$ ) from the root to a leaf is computed by one call of the temporally contingent planner. This leads to a contradiction when connecting the branches. Therefore TraCE either returns a complete temporally-contingent plan or failure.

In our approach, we analyse the plan size considering the number of actions in the plan. Other metrics such as the plan's makespan are not suitable for analysing the completeness in this case as we can find branches (described by a deterministic plan solution) with a small number of actions; however, with a higher makespan cost. For instance, consider  $\Pi_{TC}^n$  denotes the solution for a  $n$  robotic planning problem. The  $b_1$  and  $b_2$  represent two branches in the plan with plan size 3 and 6, respectively. The branches' makespans are 40 min and 20 min, respectively. The first plan described by  $b_1$  includes a navigation action with a significant duration. In this case, there is not a necessary correspondence between the two metrics. In our problems the makespan is ill-situated compare to the plan size in order to analyse the plan's branches. Following this reasoning, the plan size metric ensures we expand the tree from the most complex branches (based on the depth) to the simple ones, ensuring a solution where no further tree expansions are possible.

**Plan Correctness.** When planning with temporal and contingent conditions, plan correctness relies on checking (i) the temporal plans that build the branches are correct, (ii) the temporally-contingent plan satisfies all the goals, and (iii) the plan has sufficient knowledge at every point that supports its execution. The step (see Algorithm 1) in the TraCE algorithm *CheckExistence* satisfies the first criterion considering FINDTEMPORALPLAN returns a correct plan. The other two conditions are associated with checking the tree expansion (see Algorithm 3). If all tasks in  $Q$  are expanded, all branches were explored. The solution only exists if the approach successfully evaluates all nodes in the tree, where no nodes precede its parents and children nodes associated with positive observations come first. If  $Q$  is empty the plan solution contains all the necessary knowledge to execute the sensing actions properly.

<sup>4</sup>Connection is guaranteed by  $H$  and  $Q$ .