Efficiently Reasoning with Interval Constraints in Forward Search Planning

Amanda Coles,[*] Andrew Coles,[*] Moises Martinez,[*] Emre Savas,[*]
Juan Manuel Delfa,[**] Tomás de la Rosa,[†] Yolanda E-Martín,[†] Angel García-Olaya[†]
[*]Department of Informatics, King's College London firstname.lastname@kcl.ac.uk
[**]European Space Agency, Oxford, UK Juan.Delfa@esa.int
[†]Department of Computer Science, Universidad Carlos III de Madrid {trosa,yescuder,agolaya}@inf.uc3m.es

Abstract

In this paper we present techniques for reasoning natively with quantitative/qualitative interval constraints in state-based PDDL planners. While these are considered important in modeling and solving problems in timeline based planners; reasoning with these in PDDL planners has seen relatively little attention, yet is a crucial step towards making PDDL planners applicable in real-world scenarios, such as space missions. Our main contribution is to extend the planner OPTIC to reason natively with Allen interval constraints. We show that our approach outperforms both MTP, the only PDDL planner capable of handling similar constraints and a compilation to PDDL 2.1, by an order of magnitude. We go on to present initial results indicating that our approach is competitive with a timeline based planner on a Mars rover domain, showing the potential of PDDL planners in this setting.

1 Introduction

Interval constraints are fundamental to modeling real-world planning problems, allowing specification of constraints on the temporal relationships between the execution of actions and the achievement of facts. For example, a Mars Rover must take a picture *during* a period in which the rover is stationary at the location, and remains so for some specified time before and after the image is taken. Such constraints appear frequently in applications, not only in space-based domains, but also in terrestrial scenarios from scheduling activities of nursing home assistance robots (Tran et al. 2017) to planning ocean liner repositioning (Tierney et al. 2012).

Due to their ubiquity in applications, reasoning with interval constraints has been a major focus of research in timeline-based planning systems (Cesta et al. 2012; Frank and Jonsson 2003; Chien et al. 2000). The need for expressive languages has been noted (Cushing et al. 2007), but handling such constraints in PDDL planners has seen relatively little attention. This paper is the first to bridge the gap between the two major paradigms in temporal planning, combining their strengths: the expressive temporal reasoning of timeline planners and the powerful heuristics of PDDL state-based planners, that are not readily accessible to timeline planners, which typically use less-scalable partial-order planning approaches. This provides a new, more efficient approach for reasoning with these problems; and paves the way for the use of decades of research on PDDL planning in space applications typically dominated by timeline planners.

The lack of attention to explicitly reasoning with interval constraints in PDDL has been, in part, due to the fact that previous work noted they can be represented in PDDL 2.1 by adding additional actions to the domain (Fox, Long, and Halsey 2004; Smith, Frank, and Cushing 2008). Early critics argued that this type of approach is cumbersome, forcing a planner to do additional work (Smith 2003); and indeed, as the first to fully define, and empirically evaluate the performance of, such a compilation in this paper, we find its scalability is very poor.

Our main contribution is a novel approach for reasoning directly with interval constraints in state-based PDDL planning. We extend PDDL to provide new mechanisms for representing interval constraints explicitly, and then go on to show how an expressive PDDL planner, OPTIC, can be extended to reason directly with these constraints. We evaluate the performance of our new approach against that of a compilation on a range of benchmark domains with interval constraints, showing empirically that it outperforms this compilation by an order of magnitude (sometimes several). Finally, to illustrate the potential of our work to in space applications, typically dominated by timeline planners, we provide an initial comparison to an APSI timeline planning system (Cesta et al. 2012), on an existing ESA Mars rover domain, originally written in DDL: the representation language for APSI. We observe that our planner performs better than APSI on many of the problem instances.

2 Background

2.1 PDDL Temporal Planning Problems

The basis for PDDL2.1 (Fox and Long 2003) temporal planning is a collection of propositions P, and a vector of numeric variables $\mathbf{v}$. These are manipulated by actions whose executability is determined a precondition: a conjunction of zero or more conditions. A *condition* is either a single proposition $p \in P$, $\neg p$, or a numeric constraint over $\mathbf{v}$. We assume numeric constraints are linear, i.e. can be written $\mathbf{w}.\mathbf{v}\{>, \geq, <, \leq, =\}c$ (elements of $\mathbf{w}$, and c, are constants).

Each durative action A has three sets of preconditions: $\text{pre}_{\vdash}A$, $\text{pre}_{\leftrightarrow}A$, $\text{pre}_{\dashv}A$. These represent the conditions that

Constraint	Satisfied *iff*
synchronize g {after [lb,ub] f}	$\forall j \cdot change(j,g), \exists i : 0 \leq i \leq j \cdot change(i,\neg f) \wedge lb \leq t_j - t_i \leq ub$
synchronize f {before [lb,ub] g}	$\forall i \cdot change(i,\neg f), \exists j : j \geq i \cdot change(j,g) \wedge lb \leq t_j - t_i \leq ub$
synchronize f {overlaps [lb,ub] g}	$\forall j \cdot change(j,\neg f), \exists i : 0 \leq i \leq j \cdot cholds(i,j,g)$ $\wedge (lb \leq t_j - t_i \leq ub)$
synchronize f {during [sl,su] [el,eu] g} (where $eu < \infty$)	$\forall j,k \cdot choldsc(j,k,f), \exists i : 0 \leq i \leq j \cdot \exists l : l \geq k \cdot choldsc(i,l,g)$ $\wedge (sl \leq t_j - t_i \leq su) \wedge (el \leq t_l - t_k \leq eu)$
synchronize f {during [sl,su] [el,∞] g}	$\forall j,k \cdot choldsc(j,k,f), \exists i : 0 \leq i \leq j \cdot$ $(cholds(i,n,g) \wedge (sl \leq t_j - t_i \leq su))$
synchronize g {contains [sl,su] [el,eu] f} (where $eu < \infty$)	$(\forall i,l \cdot choldsc(i,l,g), \exists j : i \leq j \leq l \cdot \exists k : j \leq k \leq l \cdot$ $choldsc(j,k,f) \wedge (sl \leq t_j - t_i \leq su) \wedge (el \leq t_l - t_k \leq eu))$
synchronize g {contains [sl,su] [el,∞] f}	$(\forall i \cdot cholds(i,n,g), \exists j : i \leq j \leq n \cdot \exists k : j \leq k \leq n \cdot$ $choldsc(j,k,f) \wedge (sl \leq t_j - t_i \leq su)$

Table 1: Interval Constraint Semantics, w.r.t. a state trajectory $\langle (S_0,0),(S_1,t_1),...,(S_n,t_n)\rangle$ induced by snap-actions $[a_i..a_n]$. The conversion of each interval cosntraint to orderings on interval start ($\vdash$) and end ($\dashv$) points is shown below it in brackets.

must hold at its start, throughout its execution (invariants), and at its end, respectively. Instantaneous effects can occur at the start or end of A: $\text{eff}^+_\vdash A$ ($\text{eff}^-_\vdash A$) denote propositions added (resp. deleted) at the start; $\text{eff}^{num}_\vdash A$ denotes any numeric effects. Similarly, $\text{eff}^+_\dashv A$, $\text{eff}^-_\dashv$ and $\text{eff}^{num}_\dashv$ record effects at the end. All numerical effects are of the form $v\{\mathbf{+=,-=,=}\}\mathbf{w.v} + c$ ($v \in \mathbf{v}$). The values of effects become available small amount of time, ϵ, after they occur.

Finally, each action A has a duration constraint: a conjunction of numeric constraints applied to a special variable dur_A denoting its duration. As a special case, *instantaneous* actions have duration ϵ, and only one set of preconditions pre A and effects $\text{eff}^+ A$, $\text{eff}^- A$, and $\text{eff}^{num} A$. A durative action A can be split into two instantaneous *snap*-actions, $A_\vdash$ and $A_\dashv$, representing the start and end of A respectively. $A_\vdash$ has precondition $\text{pre}_\vdash A$ and effects $\text{eff}^+_\vdash A$, $\text{eff}^-_\vdash A$, $\text{eff}^{num}_\vdash A$. $A_\dashv$ is the analogous action for the end of A.

A solution is a sequence of timestamped actions, with associated durations, that transforms the initial state I into a state that satisfies G; such that the start and end preconditions of all actions are satisfied at the time they start/end; all invariant conditions hold throughout the duration of each action, and all temporal/duration constraints are respected.

2.2 Temporal Timeline-Based Planning

While timeline planners diverge in a number of ways they share common concepts, and their divergence is not crucial to the content of this paper. As we are working on a project related to the European Space Agency we use, without loss of generality, their APSI nomenclature and an APSI planner.

A timeline planning task comprises a domain and problem. The domain contains a set of components C and synchronizations S among them. There are two main types of component: state variables, that model discrete subsystems (e.g. a camera) described by a set of states V and transitions among them; and (typically bounded) numeric variables, used to represent resources (e.g. memory). A timeline is a set of non-overlapping temporal intervals with associated values of one single component. The timeline must be complete, i.e. it must define the value $v_i \in V$ of the component $c_i \in C$ at any given time $t \in [o,h]$, where $[o,h]$ is the interval of the problem defined by its origin and horizon. Components and timelines have no equivalent representation in PDDL, but it is possible to post-process a PDDL plan to generate a timeline plan (Ocon et al. 2017).

```
SYNCHRONIZE Camera.camera{
 VALUE TakingPicture(?file_id1, ?x1, ?y1, ?pan1, ?tilt1){
  cd2 <!> RobotBase.robot_base.At(?x2, ?y2);
  cd3 <!> Platine.platine.PointingAt(?pan2, ?tilt2);
  DURING [5, +INF] [0, +INF] cd2;
  DURING [2, +INF] [0, +INF] cd3;
  cd2 CONTAINS [0,+INF] [0,+INF] cd3;
  ?x1 = ?x2; ?y1 = ?y2; ?pan1 = ?pan2; ?tilt1 = ?tilt2;}}
g1 <goal> Camera.camera.TakePicture(0,0,file_1);
g2 <goal> Camera.camera.TakePicture(30,20,file_2);
g1 BEFORE g2;
```

Figure 1: (a)bove: Example Synchronize Block in DDL; (b)elow: Example of temporal relation between goals

Fundamental to timeline planning is the notion of a time interval T, represented as two timepoints $\langle T_s,T_e\rangle$ denoting its start and end. A constraint between two intervals X and Y relates the start or end of X with the start or end of Y. A synchronization s over a value v_i is used to define the set of interval constraints that must be satisfied in order to add v_i to the plan. For example, in Figure 1(a) three temporal constraints are defined: during the time the picture is taken, the rover must stay in position; during the time the picture is taken, the platine (camera mast) must be pointing at the target; and the time at which the rover is in position, must contain the time during which the platine is pointing at the target. Note the synchronization over the value v_i only applies when that value holds: in our example, for the camera to take the value TakingPicture(...) the conditions on the rover and platine must be satisfied; but, we are otherwise free to move the rover or platine without taking a picture. Synchronizations are the analog of (pre)conditions and effects of PDDL actions, but with a richer temporal vocabulary.

A DDL problem (Cesta and Oddi 1996; 2004; 2008), as in PDDL, defines a list of facts and goals. The facts define the initial state of each timeline and the list of events (analogous to timed-initial literals) that are not controllable (e.g. a com-

munication window). The goals define a list of values for different timelines to be satisfied at some point within $[o, h]$. It is possible to define an interval in the plan at which each goal must be true and to define temporal relations over the goals, providing a partial order between them, as shown in Figure 1(b): as the plan must satisfy the goals these relations must be satisfied at least once in any valid plan.

Two intervals X and Y can be related in 13 possible ways: 7 cases plus their inverses (equals has no inverse). Timeline planners such as APSI (Cesta et al. 2012), Europa (Frank and Jonsson 2003) and ASPEN (Chien et al. 2000) provide a variety of qualitative (`A EQUALS/MEETS/STARTS/ENDS B`) and quantitative temporal relations (e.g `A BEFORE [lb,ub] B`) between actions and/or states, thus effectively combining Allen's interval algebra (Allen 1983) and quantitative temporal expressions (Dechter, Meiri, and Pearl 1991). A quantitative relation between X and Y generalizes qualitative relations by limiting the lower bound (lb)/upper bound (ub) distance between their endpoints. In this work we support all 13 Allen constraints, for brevity we focus on the quantitative constraints listed in Table 1; the reasoning for qualitative constraints can trivially be derived by setting $ub/lb/sl/su = 0$ or ∞ as appropriate in the equivalent quantitative constraint.

As the first step of translating a DDL model to PDDL we introduce the notion of a 'state holding' durative action. This represents the transition of a DDL component c_j to, and then from, a value $v_i \in V$. For each component c_j that can take value v_i we create an action A, with associated fact $c_j v_i$, that regulates the transition of $c_j v_i$ from false, to true, and back again: pre $A_{\vdash} = \{\neg c_j v_i\}$, eff^{+} $A_{\vdash} = \{c_j v_i\}$, pre$_{\leftrightarrow} A = \{c_j v_i\}$, eff^{-} $A_{\dashv} = \{c_j v_i\}$. The challenge remains to enforce temporal constraints between state-holding durative actions for different DDL components with respect to each other to respect the DDL synchronization constraints (S). This reduces to enforcing interval constraints between starts and ends of PDDL durative actions, the topic of the rest of the paper.

2.3 Interval Constraints vs PDDL3 Constraints

The ability to express constraints and preferences over the trajectory of the plan was introduced in PDDL3 (Gerevini et al. 2009) and a number of planners were developed to support these (Benton, Coles, and Coles 2012; Edelkamp, Jabbar, and Nazih 2006; Baier, Bachus, and McIlraith 2007). Two major semantic differences make PDDL 3 constraints insufficient to capture Allen constraints. The first is change semantics: DDL constraints are defined w.r.t. steps at which the truth value of a fact (or formula) *changes*; whereas PDDL3 constraints are defined w.r.t. whether the truth value of a fact (or formula) *holds* at a step. Consider the DDL constraint `synchronize A {BEFORE [0,10] B}`. To capture this, one might propose state-holding durative-actions A and B, with associated facts a and b, and a PDDL constraint `(always-within 10 (a) (b))` ($\forall i \cdot Si \models a, \exists j : i \leq j \leq n \cdot Sj \models b \wedge t_j - t_i \leq 10$). The plan `0:(A) [5], 1:(B) [10]` (i.e. start B as soon as A starts) satisfies this PDDL3 constraint, but does not satisfy the DDL constraint. `(always-within 10 (not (a)) (b))` does not help either: it does require B to start after A ends, but as soon as B ends, a state is reached where $\neg a \wedge \neg b$ is true, so b must be achieved again within 10 time units.

Second, we are not able to refer to a specific instance of a fact becoming true in PDDL so we cannot define transitive relationships: for example, in DDL if a synchronization defines `A BEFORE [0,INF] B` and `B BEFORE [0,INF] C` the same A,B and C must be used for both, thus enforcing A, B, C. But, `(and (sometime-after (not (a)) (b)) (sometime-after (not (b)) (c)))` can be satisfied by $\neg b$, c, $\neg a$, b ($B_{\dashv}$,$C_{\vdash}$,$A_{\dashv}$,$B_{\vdash}$). Finally, we note that PDDL 3 constraints lack the ability to express lower bounds.

2.4 Other Related Work

There have been a number of other approaches to reasoning with interval constraints in planning. Perhaps the most closely related is the planner MTP (To et al. 2017; 2016), it searches in a space of world models that use a timeline-inspired representation, and makes use of a SAT solver and Model Checker, which relies on a discretisation of time, to ensure constraints are satisfied. Since MTP uses a PDDL-style representation we are able to compare to it in our evaluation. The ANML language was developed to act as a bridge between PDDL and the more traditional timeline-based NASA languages, in particular NDDL (Bedrax-Weiss et al. 2005), used by Europa2, and AML (Sherwood et al. 2005) used by Aspen (Chien et al. 2000). It was written to provide a language that could be used for both HTN-based and generative planning and allows the expression of interval constraints similar to those we have here. The planner FAPE (Dvorak et al. 2014) was developed to reason with the ANML language, and is capable of reasoning with interval constraints such as those considered in our paper. It performs plan-space planning, which is traditionally less efficient than forward-search, but does support HTN decomopostition to improve efficiency. A fair empirical comparison to FAPE is not possible here, due to the difference in representation language, and the HTN planning approach. Finally, we note that other researchers (Gigante et al. 2016) have shown that timeline based languages are able to express PDDL problems, which is the complement of our work.

3 Reasoning with Interval Constraints

In this section we discuss how interval constraints can be handled natively in a forward-search temporal planner.

3.1 State Consistency Checking in Optic

We build on the forward-search planner OPTIC (Benton, Coles, and Coles 2012). Search begins from the initial state S_0, and successive application of snap-actions $[a_1..a_n]$ yields a state-trajectory $[S_1..S_n]$. Plan steps are partially ordered according to the facts they refer to. To facilitate this, each fact p, in each state, is annotated with:

- $F^{+}(p)$ ($F^{-}(p)$): the index of the plan step that most recently added (deleted) p;
- $FP^{+}(p)$: a set of pairs, each $\langle i, d \rangle$, used to record steps with a precondition p. i denotes the index of a plan step, and $d \in \{0, \epsilon\}$. If d=0, then p can be deleted at or after

step i: this corresponds to the end of an invariant condition. If d=ϵ, then p can be deleted ϵ after i or later.

Applying actions to states produces ordering constraints based on the annotations and updates their values. These ordering constraints form a simple temporal problem (STP):

- Steps adding (deleting) p are ordered ϵ after $F^-(p)$ ($F^+(p)$), ensuring effects on a fact are totally ordered. Preconditions are fixed within this ordering: a step with precondition p is ordered after $F^+(p)$; recording it in $FP^+(p)$ ensures future deletors of p are ordered after it.
- If step j ends an action A that began at step i, the interval $[i, j]$ must respect the duration constraints of A.

A similar set of annotations and update rules is used for numeric variables: the index of the last effect on each variable is recorded; the indices of steps with preconditions on the variable (or effects referring to its value) are recorded; and ordering constraints are generated from these.

In this work, we build on the Mixed Integer Program (MIP) approach OPTIC uses to support PDDL3 (Gerevini et al. 2009) trajectory constraints, we inherit the OPTIC'S standard search and memoization techniques (Coles and Coles 2016). OPTIC'S MIP, solved at each state during search, has a continuous variable t_i representing the time of each plan step with index i. These steps are constrained according to the STP, to ensure the plan is temporally consistent: for each ordering constraint imposed during search we have $t_j - t_i \geq \epsilon$ (or 0); and for each durative action starting at step i and finishing at step j we write the duration constraints over t_i and t_j e.g. $t_j - t_i \geq min_{durA}$ and $t_j - t_i \leq max_{durA}$.

To capture trajectory constraints, binary decision variables and associated constraints were introduced – there may be disjunctive choice over how exactly to satisfy these constraints. The MIP finds an assignment of values to each t_i (i.e. timestamps for the start and end points of the actions in the plan) that satisfies the constraints; or if no solution can be found, the constraints cannot be satisfied, so the state is pruned. In contrast to the optimisation of soft constraints in OPTIC, in this work we are interested in finding satisfying solutions that respect hard constraints on time intervals; we are not optimising a particular objective.

3.2 Modeling Interval Constraints

Table 1 defines interval constraints in terms similar to those used to define PDDL 3 constraints (Gerevini et al. 2009). We use the following concepts defined over the state trajectory $\langle (S_0, 0), (S_1, t_1), ..., (S_n, t_n) \rangle$ produced by executing a plan of snap-actions $[a_1..a_n]$:

$$change(i, f) \Leftrightarrow \begin{array}{ll} (S_i \models f \wedge S_{i-1} \models \neg f) & \text{if } i \geq 1 \\ (S_0 \models f) & \text{otherwise} \end{array}$$

$$\begin{array}{rl} holds(i, j, f) \Leftrightarrow & \forall k : i \leq k \leq j, S_k \models f \\ cholds(i, j, f) \Leftrightarrow & (change(i, f) \wedge holds(i, j, f)) \\ holdsc(i, j, f) \Leftrightarrow & (holds(i, j\text{-}1, f) \wedge change(j, \neg f)) \\ choldsc(i, j, f) \Leftrightarrow & (change(i, f) \wedge holdsc(i, j, f)) \end{array}$$

We now define PDDL analogs for DDL interval constraints. As discussed earlier a synchronization such as Figure 1 maps to a PDDL durative action. Thus, we allow durative actions to have their own `:constraints` section, where:

- `(interval A (pred <?params>))`: A denotes a pair of plan steps ⟨As,Ae⟩ where *choldsc* (As,Ae,`(pred <?params>)`). For instance, `(interval cd3 (Platine.platine.PointingAt ?pan2 ?tilt2))` corresponds to the definition of `cd3` from Figure 1.
- `(= ?p ?q)` denotes that parameters p and q (parameters of the action or of an interval defined therein) are equal.
- for each DDL constraint `X`, we create a new PDDL keyword `constrain-X`. As in DDL, these then impose constraints on the defined intervals, or a special interval `this` referring to the action itself. For instance, `(constrain-DURING this 5 inf 0 inf cd3)` and `(constrain-CONTAINS cd2 0 inf 0 inf cd3)` correspond to the constraints on `cd3` from Figure 1.

We also allow each of these to appear in the :constraints section of a problem, to specify temporal constraints over goals.

3.3 Enforcing Temporal Constraints in the MIP

We now detail how we extend the MIP in OPTIC to generate plans that satisfy interval constraints. Recall that the MIP in OPTIC contains a variable t_i representing the time at which the snap-action at index i in the plan is applied, which is constrained according to the duration constraints of actions and the ordering constraints it generates during search. A MIP solution assigns values to t_i to satisfy the temporal (and any other) constraints. Our approach extends this MIP with constraints to also enforce interval constraints by encoding the quantified formulæ defined in Table 1 over the state trajectory produced by the plan.

Our MIP must address three challenges to enforce interval constraints. First, it must determine *which* pairs of steps represent valid start and end points for intervals. If action B, starting at step Bs and ending at step Be, has `(interval X (f))` in its `:constraints` section, there may be many points in the plan where f changed from false to true (or true to false), hence several candidate pairs of steps Xs and Xe, that could represent the start and end of the interval X. Second, once we have identified the candidate pairs (Xs, Xe) the MIP must select which of these will be chosen to be the interval X (we cannot enforce the interval constraints over an arbitrarily chosen pair (Xs, Xe) as there may not exist a solution plan with that pair satisfying the constraints; but there may exist another pair that does admit a solution). Finally, given this choice of (Xs, Xe) we must enforce all temporal constraints defined for Bs, over these. For example, if we also have at Bs `(interval Y (g))` `(constrain-DURING X sl su el eu Y)` we must add constraints to ensure the chosen Xs,Xe and Ys,Ye satisfy this.

Marking Candidate Intervals in the Plan DDL constraints are based on points in the plan at which a formula *changes* from false to true, or vice versa. These change points mark the valid start and end point pairs that we need to mark as possible candidates for representing intervals referring to that formula in the plan. In order to mark these we introduce dummy steps to record their position in the plan.

During search, after the application of a snap-action a_i to yield a state S_i, we evaluate the truth value of all formulæ

f seen in any interval constraint whose truth value might conceivably have changed compared to S_{i-1}; that is, where the effects of a_i modify propositions and variables referred to in f. If $S_i \models f$, (and $S_{i-1} \models \neg f$) we immediately add to the plan a dummy step $o_i(f)$ with condition f, and no effects; likewise, if $S_i \models \neg f$ (and $S_{i-1} \models f$) , we add a dummy step $o_i(\neg f)$ with precondition $\neg f$, and no effects. These dummy steps serve to generate ordering constraints, and update annotations, to assert the truth value of a formula.

Unlike applying snap-actions, which have successive step indices, the dummy step shares the step index of the snap action i: this ensures i is ordered at least 0 time units after the last step to affect any proposition $p \in f$, or variable $v \in f$; and the annotations in S_i are updated to ensure future effects on p (resp. v) are ordered at least 0 after i. Binding $o_i(f)$ (or $o_i(\neg f)$) to step i ensures that t_i marks the exact point some snap-action a_i was applied, and changed the state into one in which f (resp. $\neg f$) holds. This differs from the approach taken in OPTIC, where dummy steps were not tied to a specific plan step: as interval constraints can have lower and upper time bounds, we cannot allow them to 'slip' from the step which caused the truth value of f to change.

In the simple case, f is a single fact that is only ever manipulated by an action that includes $\neg f$ in its start preconditions; f in its start effects; and $\neg f$ in its end effects. This is analogous to f being a fact that is true *iff* a DDL variable takes the value f: $o_i(f)$, $o_i(\neg(f))$ mark when the variable transitioned into and out of this value, defining the interval relative to which interval relations are defined.

Returning to our first challenge, recall our canonical action B, starting at step Bs and ending at step Be, with `(interval X (f))` in its `:constraints` section, candidate steps for Xs, are now simply all steps at which $o(f)$ occurred in a plan: $O(f,\pi) = \{i \cdot 0 \leq i \leq n \wedge o_i(f) \in \pi\}$. The second challenge requires us to encode the choice over these in the MIP. We do this as a set of decision variables: for each $i \in O(f,\pi)$ the variable $Bs_{Xs=i} \in \{0,1\}$ indicates that step i was chosen as the start of interval X for the action that started at step Bs. Likewise, for each $j \in O(\neg f,\pi)$, the variable $Bs_{Xe=j} \in \{0,1\}$ indicates that step j was chosen as the end of interval X for this step.

It may be that, depending on the interval constraint used, one of Xs or Xe is relevant. For instance, if B is constrained to come before X, only Xs is relevant, and Xe need not necessarily be defined (in which case, it does not matter if $O(\neg f,\pi)$ is empty). Hence, we do not insist that the choice is made, unless a constraint requires it. Regardless, if steps i, j are chosen as Xs, Xe, then we need to ensure that j was the first step at which $o(\neg f)$ occurred after i:

$$Bs_{Xs=i} + Bs_{Xe=j'} \leq 1 \quad \forall i \in O(f,\pi), \; \forall j' > \min[j \in O(\neg f,\pi) \cdot j > i]$$

Enforcing Interval Constraints on Plans We now come to the final challenge, writing the interval constraints at Bs as MIP constraints over the chosen decision variables. We must take care to only enforce each constraint between the plan steps chosen as the interval start/end points. For `(constrain-BEFORE X lb ub Y)` this is the step chosen to be the end of X, and that chosen to be the start of Y; i.e. the pair of steps i, j where $Bs_{Xe=i}$ and $Bs_{Ys=j}$ are set to 1. We use "big-M" constraints for this, M is a large constant comfortably exceeding the duration of the solution plan. For $lb \leq t_{Ys} - t_{Xe} \leq ub$ constraining the end of X and the start of Y–defined for the action B starting at step Bs–we write:

$$t_j - t_i + M(2 - Bs_{Xe=i} - Bs_{Ys=j}) \geq lb \qquad \forall i,j \qquad (1)$$
$$t_j - t_i - M(2 - Bs_{Xe=i} - Bs_{Ys=j}) \leq ub \qquad \forall i,j \qquad (2)$$

Intuitively, if step i is chosen to be the end of X, and step j is chosen to be the start of Y, then $Bs_{Xe=i} + Bs_{Ys=j} = 2$ and the big-M term disappears – enforcing the temporal constraints. For any other setting of the decision variables, the big-M makes the constraints trivial: they will be satisfied regardless of the values of t_i and t_j. To ensure a choice of i and j is definitely made, we further write:

$$\textstyle\sum_i Bs_{Xe=i} = 1 \wedge \sum_j Bs_{Ys=j} = 1 \qquad (3)$$

Now we must ensure that $i \leq j$: as OPTIC searches forwards, steps added later in the plan never have ordering constraints placing them before earlier steps. So, $Bs_{Xe=i}$ can only be set to 1, if a later (higher index) $Bs_{Ys=j}$ is set to 1:

$$Bs_{Xe=i} \leq \textstyle\sum_{j \cdot j \geq i} Bs_{Ys=j} \qquad \forall i \qquad (4)$$

We can enforce all interval constraints defined for the action starting at Bs by creating copies of equations 1–4 for each one. The given equations show how we constrain $Bs_{Xe=i}$ and $Bs_{Ys=j}$ to enforce a before constraint but all other constraints we consider can be represented by writing equations 1–4 over different pairs of end points and setting $[lb, ub]$ to the appropriate bounds. Table 2 shows the pairs of variables to respectively replace $Bs_{Xe=i}$ and $Bs_{Ys=i}$ with equations in 1–4 to enforce all of of our quantitative constraints (the corresponding qualititative constraints can be modelled by setting $ub = lb = 0$).

Constraint	Replace $Bs_{Xe=i}$	Replace $Bs_{Ys=j}$	bounds lb	ub
`(AFTER X lb ub Y)`	$Bs_{Ye=i}$	$Bs_{Xs=j}$	lb	ub
`(OVERLAPS X lb ub Y)`	$Bs_{Ys=i}$	$Bs_{Xe=j}$	lb	ub
`(DURING X sl su el eu Y)`	$Bs_{Ys=i}$	$Bs_{Xs=j}$	sl	su
and repeat equations 1–4 with	$Bs_{Xe=i}$	$Bs_{Ye=j}$	el	eu
`(CONTAINS X sl su el eu Y)`	$Bs_{Xs=i}$	$Bs_{Ys=j}$	sl	su
and repeat equations 1–4 with	$Bs_{Ye=i}$	$Xs_{Ye=j}$	el	eu

Table 2: Endpoints to constrain to enforce interval constraints showing substitutions to make in Equations 1–4.

Finally we consider constraints relative `this`, i.e. the action starting at Bs itself. We know that Bs and Be are the appropriate start–end steps for `this`, so do not need to make a decision over this. Thus, after creating decision variables for the interval `this` relative to Bs, we fix its position:

$$Bs_{THISs=Bs} = 1 \qquad Bs_{THISe=Be} = 1$$

We note that this approach maintains a great deal of execution flexibility. While steps and their orderings are defined during search, timestamps for their execution are only set when the MIP is solved for the solution plan. Moreover, at execution time, if step i is delayed we can solve the MIP again with t_i set to the actual execution time, to obtain a new valid schedule for the plan (if one exists).

Enforcing Interval Constraints on Incomplete Plans As in OPTIC, we must define a relaxation of this MIP to be used

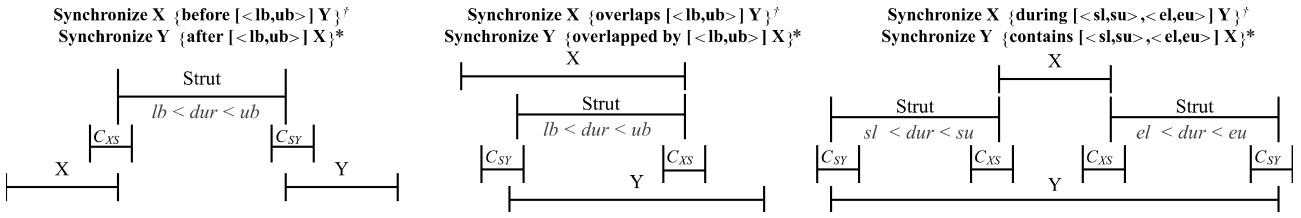

Figure 2: Compilation of Quantitative Constraints. For† C_{AB} restricts A and requires B; for* C_{AB} requires A and restricts B.

during search: we cannot insist that a partial plan to some state meets all interval constraints, as it may be the case that adding more actions to the plan would satisfy them.

Consider again an action B, starting at step Bs and ending at step Be, that defines an interval X; and interval constraints between this and X. If these imply an ordering constraint $l1 \leq t_{Bs} - t_{Xs} \leq u1$ (or $l2 \leq t_{Bs} - t_{Xe} \leq u2$), an interval X must already have started (or ended) in the plan, prior to Bs. Thus, constraints of this form cannot be relaxed for partial plans: actions in any extension of the plan cannot precede Bs, so we add the requisite decision variables and big-M constraints, as per equations 1-4. Conversely if these imply an ordering constraint $l1 \leq t_{Xs} - t_{Be} \leq u1$ we can relax these constraints (omit them from the MIP) as actions could be added later to the plan to satisfy them.

If this relaxed MIP cannot be solved, no extension of the partial plan is valid: if it was extended to be a candidate solution plan, then when encoding the interval constraints for the start/end of B at steps Bs/Be, the only options for Xs/Xe that are available to precede the start/end of B must already be in the plan before these steps. Thus, it is completeness-preserving to insist the partial plan meets these constraints.

4 Compiling Interval Constraints into PDDL

As noted earlier, we can enforce all DDL temporal relations by constraining the time between the starts and ends of snap-actions. To compile this we can use *clips* (Fox, Long, and Halsey 2004). With reference to Figure 3 (left), each clip has one or more pair(s) of start and end preconditions that *require* one or more snap actions to be applied during the clip (here, $x \in \mathit{eff}^+_{\vdash}(X)$ and $\mathit{eff}^-_{\dashv}(X)$); and a pair of effects (here C_{XY} and $\neg C_{XY}$) that *restrict* some snap action(s) (to which C_{XY} is added as a precondition) so that it can only be applied during the clip. When a synchronization places a constraint between X and Y, but not with itself – or if X and Y are goals subject to a constraint – then both X and Y must happen at least once, and there must be one pair X,Y that satisfies the constraint. In this case we use a clip that *requires* both X and Y but *restricts* neither, and additionally adds a fact at the end (e.g. met_{XY}) to represent that the constraint is satisfied: this fact becomes a goal of the PDDL problem, or an end-condition of the action corresponding to the synchronization, as applicable.

Figure 3 (left), demonstrates how we can enforce `synchronize X {MET BY Y}` (for Y to start, X must end) using a clip that *requires X* and *restricts Y*. Soundness follows from the fact that Y cannot be executed outside the clip (due to precondition C_{XY}) and the clip cannot execute unless X ends within it (due to preconditions x (start) and $\neg x$ (end)). If we want to enforce the inverse relationship, `synchronize X {MEETS Y}`, we instead make the clip *restrict* $X_{\dashv}$ and *require* $Y_{\vdash}$. This mechanism for synchronizing two snap actions allows us capture to all of the qualitative Allen Relations: `X MEETS Y` clips $X_{\dashv}$ to $Y_{\vdash}$; `X STARTS Y` clips $X_{\vdash}$ to $Y_{\vdash}$; and `X ENDS Y` clips $X_{\dashv}$ to $Y_{\dashv}$. Inverse relations are obtained by swapping $X_{\dashv/\vdash}$ with $Y_{\vdash/\dashv}$.If the condition with respect to Y appears inside `synchronize X{...}`, then we use a clip that *requires* Y and *restricts* X.

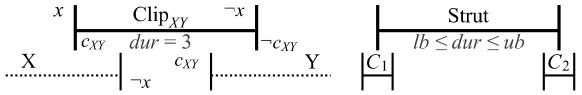

Figure 3: Compilation Components (note Clip_{XY}, c_{XY} and Strut are parameterised to make a unique clip/strut for each pair of ground actions, parameters omitted here for clarity).

To model quantitative constraints we use the structure in Figure 3 (right) comprising two clips and a *strut*. Our strut is inspired by Halsey *et al.* (2004) but we add a flexible duration ($lb \leq \text{dur} \leq ub$), allowing us to enforce both lower and upper bounds on the interval between two snap actions using one action. The start/end of the strut are respectively clipped to snap actions A_1 and A_2. C_1 *requires* A_1 and *restricts* $\text{strut}_{\vdash}$; C_2 *requires* $\text{strut}_{\dashv}$ and *restricts* A_2. Hence, A_2 can only occur $[lb, ub]$ after A_1: A_2 can only be applied if clipped to $\text{strut}_{\dashv}$; and $\text{strut}_{\vdash}$had to be clipped to A_1.

Figure 2 shows the compilation of quantitative constraints. The constraint enforced is determined by the clips as for qualitative constraints. Clips that require X and Y but restrict neither are used if a synchronization places a constraint between X and Y, but not with itself; or if X and Y are goals. If a constraint appears in a synchronize block for X or Y itself (e.g. `sync X {BEFORE [lb,ub] Y}`) the clips *require/restrict* X and Y as described in the caption. The compilation can streamlined when lb=0 or ub=∞, e.g. `sync X {BEFORE [0,INF] Y}`, it suffices to add a dummy goal that is true initially, deleted by $X_{\dashv}$ and added by $Y_{\vdash}$.

5 Evaluation

In this section we evaluate our approach in two ways. We compare the performance of our native approach to the compilation and to the MTP planner across a range of PDDL benchmark instances with interval constraints added. We then compare both approaches to the default planner provided with APSI 3.3.2 on a Mars rover domain originally

Domain	Ver	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Rover	nat	0.17	0.76	5.47	19.97	58.98	151	0.71	1.13	3.21	32.89	79.44	203.4	0.38	0.55	1.79	4.72	10.94	22.35	x	x
Rover	comp	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x
Rover	APSI	2.22	2.36	2.43	2.56	2.76	2.86	3.1	5.4	16.7	327.6	-	-	3.69	19.62	-	-	-	-	x	x
Rover	MTP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x
Concrete	nat	0.01	0.01	0.01	0.13	0.14	0.59	0.43	1.72	5.07	4.36	6.3	14.37	23.29	27.48	39.22	54.19	67.45	103.8	138.2	88.48
Concrete	comp	0.15	3.22	0.16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Concrete	MTP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Zeno	nat	0.01	0.05	0.12	0.06	0.28	0.28	0.78	0.91	9.24	64.59	4.43	1.00	117.4	-	-	-	-	-	-	-
Zeno	comp	0.01	0.07	3.76	428.2	2.89	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Zeno	MTP	0.25	-	13.60	466.4	-	183.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Cafe	nat	0.01	0.01	0.01	0.02	0.03	0.05	0.03	0.04	0.04	0.24	0.5	1.53	3.54	7.12	13.23	21.8	36.75	85.86	157.0	474.1
Cafe	comp	0.06	0.12	0.12	0.26	0.5	0.85	0.5	0.85	0.86	7.92	22.1	120.1	-	-	-	-	-	-	-	-
Cafe	MTP	0.60	4.29	1.48	20.86	329.7	-	335.3	-	-	-	-	-	-	-	-	-	-	-	-	-
Crewplanning	nat	0.02	0.03	0.02	0.07	0.11	0.18	0.26	0.23	0.3	0.56	18.98	-	-	-	-	-	-	-	-	-
Crewplanning	comp	11.43	3.02	11.56	84.78	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Crewplanning	MTP	1238.1	-	1159.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 3: Time (in seconds) taken to solve problems in Domains with Interval constraints, with native handling (nat) versus the compilation (comp). '-' indicates that the problem was unsolved, 'x' that the problem does not exist.

written in DDL for an ESA project and translated to PDDL using the methods described in this paper. This allows us to show not only that we now have an efficient native mechanism for handling Allen Constraints in PDDL, but also to make the case that PDDL planners can be competitive when it comes to solving space related problems, traditionally handled using timeline based approaches.

We compared the performance of our native approach, the compilation and MTP on the following PDDL domains. For MTP, these were translated to MPDDL. **Concrete (new)**: Trucks transport concrete, maintaining the liquid state by mixing. The mixing action must occur [2,10] minutes after initially combining the ingredients, and must meet the action to offload concrete. All instances have 2 trucks and scale from delivering concrete to 2–6 sites. **Crewplanning (IPC 2008)**: A domain involving planning the daily routine of astronauts. We added constraints that astronauts must have a meal 15–300 minutes after waking up, and exercise at least 30 minutes before going to bed. **Zeno Travel (IPC 2000)**: Passengers must be transported via plane to specific goal airports. We added interval constraints to require that aircraft must leave an airport 20–30 minutes after arriving. **Cafe (Halsey 2004)**: Orders placed in a cafe must be prepared and delivered. We added a constraint that food is delivered 1–3 minutes after cooking.

The challenges posed by interval constrainst vary in our domains: in Concrete, solution plans can be padded to satisfy interval constraints; whereas in other domains constraints change the structure of solution plans, e.g. the constraint that crew must eat within an interval after waking: searching without considering interval constraints would yield plans that could not be scheduled to satisfy them.

Our results in Table 3, show that while the compilation has the expressivity to model temporal relations, it is not feasible for solving larger problems as it introduces a large number of additional actions. These increase the branching factor, and lengthen the solution plans the planner must find. Further, they make it much more likely there is a currently executing action in a state, hence memoisation pruning has to be much more conservative (Coles and Coles 2016). This overhead is incurred by the compilation even in the Concrete domain, where the interval constraints should not, in theory, pose a significant challenge to solving the problem

The native approach significantly outperforms the compilation by an order of magnitude (sometimes several). Indeed, any problem solved *at all* by the compilation is solved by the native approach in under 2 seconds; and the native approach scales significantly beyond the compilation's limits. Whilst MIP in general is NP-Hard, the MIPs generated during planning are relatively small and simple so this is not an issue for the native approach in practice: MIP building/solving remains $<$15% of planner runtime and is dwarfed by heuristic computation time as in prior work (Coles et al. 2012). We note that, while the native approach cannot handle the largest Crewplanning and Zeno problems, this is not because interval constraints hinder search; but rather that these competition problems are challenging for OPTIC even without additional constraints.

Although MTP can reason with interval constraints encoded in MPDDL, it was initially unable to solve any of the benchmark problems. This is because it relies on a predefined discretization of time that is not fine enough to admit solutions to the problems; i.e. it cannot satisfy tightly bounded interval constraints (e.g. the [1,3] interval in Cafe). We cannot change the discretization as the source code is unavailable. To give MTP a chance to solve these problems we divided all durations and temporal constraints by 10. This does not affect the performance of the native approach, or compilation but allowed MTP to solve some problems; these are the results shown for MTP in Table 3. Even with this scaling to suit its discretization, our native approach solves many more problems and solves mutually solved problems at least an order of magnitude faster than MTP. In fact, MTP is generally outperformed even by the compilation. The makespan of plans produced by MTP were often much longer, and never better, than our approach.

Our comparison to timeline based planners is focused on APSI: is was not possible to compare to other timeline planners, as these are generally commercial systems with no uniform input language. We focus on a single Mars rover

domain as there are no available standard benchmarks for timeline based planning. The rover must take pictures at certain waypoints, and periodically transmit them. The constraints on taking a picture are as Figure 1. Data transmission involves three interval relations. The rover must stay at a designated position **during** (and some time before and after) transmission by the antenna; the image must be taken a specified interval **before** it is transmitted; and the antenna can only be turned on to transmit **during** a communication window, and the rover's position must be syncronized **during** the antenna being switched on. We defined three groups of problems: the first problems (1-6) involve totally ordered goals limited to taking pictures at waypoints. The second (7-12) additionally require a communication activity for each picture; the third (13-18) are equivalent to 7-12, but with no temporal ordering between the goals. We created 6 problems in each group by increasing the number of images taken and the number of locations (both ranging from 2 to 7).

APSI is faster than OPTIC only on the simplest problems (1-6), where no communication is required, scaling better as problem size increases. In more complex problems (7-12), OPTIC clearly outperforms APSI, solving within minutes problems unsolved by APSI after 30 minutes. Interestingly, OPTIC performs better in problems 13-18 where goals are not ordered because when the total ordering is contrary to the search guidance provided by the heuristic, this leads to more of the state space being explored. Conversely, APSI performs much worse here: if a goal ordering is given, it is used to guide search. While the compilation did not solve any Rover evaluation problems, in smaller tests it could achieve a single goal of taking a picture at the initial location, or to be at another location, but not to both travel and take a picture. Again, neither planner is attempting to optimise a specific metric, but for interest, we note that both planners produce identical plans for all solved problems.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 Research and Innovation programme (Grant Agreement 730086, ERGO); EPSRC grant EP/P008410/1 (AI Planning with Continuous Non-Linear Change); the European Space Agency (ESA/ESTEC) GOTCHA project, Contract No. 4000117648/16/NL/GLC/fk; and Ministerio de Economía, Industria y Competitividad TIN2017-88476-C2-2-R and TIN2015-65686-C5.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26:11.

Baier, J.; Bachus; and McIlraith, S. 2007. A heuristic search approach to planning with temporally extended preferences. In *IJCAI*.

Bedrax-Weiss, T.; McGann, C.; Bachmann, A.; Edgington, W.; and Iatauro, M. 2005. Europa2: User and contributor guide. *Technical report, NASA Ames Research Center*.

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

Cesta, A., and Oddi, A. 1996. DDL.1: A formal description of a constraint representation language for physical domains. *New Directions in AI Planning.*

Cesta, A., and Oddi, A. 2004. Planning with concurrency, time and resources: A CSP-based approach. *Intel. Techniques for Planning.*

Cesta, A., and Oddi, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Arc. Control Sciences*.

Cesta, A.; Fratini, S.; Orlandini, A.; and Rasconi, R. 2012. Continuous planning and execution with timelines. In *International Symposium on Artificial Intelligence*.

Chien, S.; Rabideau, G. R.; Knight, R. L.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T. A.; Smith, B.; Fisher, F. W.; Barrett, A. C.; Stebbins, G.; and Tran, D. 2000. Aspen - automated planning and scheduling for space mission operations. In *Space Ops*.

Coles, A. J., and Coles, A. I. 2016. Have I Been Here Before? State Memoisation in Temporal Planning. In *Proc. ICAPS*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. COLIN: Planning with Continuous Linear Numeric Change. *JAIR* 44.

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

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

Dvorak, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and Acting with Temporal and Hierarchical Decomposition Models. In *ICTAI*, 115–121.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-Scale Optimal PDDL3 Planning with MIPS-XXL. In *IPC5 booklet, ICAPS*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *JAIR* 20.

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of PDDL2.1. In *Proceedings ECAI*.

Frank, J. D., and Jonsson, A. K. 2003. Constraint-based attribute and interval planning. *Journal of Constraints* 8:4.

Gerevini, A. E.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic Planning in the Fifth IPC: PDDL3 and Experimental Evaluation of the Planners. *AIJ*.

Gigante, N.; Montanari, A.; Mayer, M. C.; and Orlandini, A. 2016. Timelines are expressive enough to capture action-based temporal planning. In *Proc. TIME 2016*.

Ocon, J.; Delfa, J. M.; de la Rosa, T.; García, A.; and Escudero, Y. 2017. Gotcha: An autonomous controller for the space domain. In *Proc. ASTRA*.

Sherwood, R.and Engelhardt, B.; Rabideau, G.; Chien, S.; and Knight, R. 2005. Aspen user's guide. *Tech. Report D15482, JPL.*

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

Smith, D. E. 2003. The Case for Durative Actions: A Commentary on PDDL2.1. *JAIR*.

Tierney, K.; Coles, A. J.; Coles, A. I.; Kroer, C.; Britt, A.; and Jensen., R. M. 2012. Automated planning for liner shipping fleet repositioning. In *Proc. ICAPS*.

To, S.; Roberts, M.; Apker, T.; Johnson, B.; and Aha, D. 2016. Mixed propositional metric temporal logic: A new formalism for temporal planning. In *AAAI Hybrid Systems Workshop*.

To, S.; Johnson, B.; Roberts, M.; and Aha., D. 2017. A new approach to temporal planning with rich metric temporal properties. In *Proc. ICAPS*.

Tran, T. T.; Vaquero, T. S.; Nejat, G.; and Beck, J. C. 2017. Robots in retirement homes: Applying off-the-shelf planning and scheduling to a team of assistive robots. *JAIR* 58.