Using Constraint Programming for Disjunctive Scheduling in Temporal AI Planning

Adam Francis Green
Department of Informatics, King's College London, United Kingdom
Tango Hospitality Inc., Canada

J. Christopher Beck
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Amanda Coles
Department of Informatics, King's College London, United Kingdom

Abstract

We present a novel scheduling model that leverages Constraint Programming (CP) to enhance problem solving performance in Temporal Planning. Building on the established strategy of decomposing causal and temporal reasoning, our approach abstracts two common fact structures present in many Temporal Planning problems – Semaphores and Envelopes – and performs temporal reasoning in a CP-based scheduler. At each search node in a heuristic search for a temporal plan, we construct and solve a Constraint Satisfaction Problem (CSP) and integrate feedback from the CP-based scheduler to guide the causal planning search towards a solution. Through experimental analysis, we validate the impact of these advances, demonstrating a significant reduction in both the number of states searched and in search time alongside an increase in problem-solving coverage.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling; Theory of computation → Constraint and logic programming; Computing methodologies → Search methodologies

Keywords and phrases AI Planning, Temporal-Numeric Planning, Constraint Programming, Scheduling

Digital Object Identifier 10.4230/LIPIcs.CP.2024.13

1 Introduction

Temporal AI Planning is an extension of classical AI planning that includes a representation of the duration of actions and reasoning about numeric variables that change over time [9]. For such a problem, a solution is a sequence of scheduled actions, called a *plan*, which transforms the world to a desired goal state from an initial description of the world. Unlike scheduling problems such as Job-Shop Scheduling [15], but similar to most planning problems, solutions to temporal AI planning problems do not have a pre-defined set of actions to execute, but rather actions must be both selected and scheduled by a planner.

Planners designed to handle temporal problems typically take one of two approaches. *Decision-epoch* planners build a plan by adapting a classical planning approach where actions are selected and applied to a state to generate a new state, when a new action is started its effects are realised and the time of its end is added to a queue, stored in the state. The planner can then decide to apply another action at the current time or advances time until after the next action in the queue, in order to realise that action's effects. Further actions can then be applied at this time point, with the process repeated until all goals are achieved and all executing actions have completed [7, 8]. In the decision-epoch approach, the planner reasons about both the causal and the temporal aspects of the problem.

Decomposition planners separate causal and temporal reasoning [11]. The planner applies actions to states to generate an atemporal successor state, then checks the temporal

consistency of the partial plan for that state using a scheduler. If the resulting state is numerically and temporally consistent, additional actions can be applied to extend the plan. If the resulting state is not consistent, it is discarded and the planner explores the expansion of other states. This alternation between planning and scheduling continues until a goal state is achieved. Thus, the causal reasoning takes place in task planning, while the temporal reasoning is done using a sub-solver model such as a Simple Temporal Network [11] or a Linear Program [3].

Decomposition Temporal Planning lends itself to transferring other temporal and/or combinatorial structures to the sub-solver, provided that it has the ability to model and solve such structures. This paper presents two contributions to decomposition approaches in temporal AI planning.

1. We show that, through the use of a constraint-based scheduler, additional temporal and combinatorial reasoning can be transferred to the sub-solver by abstracting semaphore and envelope fact structures, reducing the number of states that the planner needs to explore.
2. We show how this transfer also enables an increase in the feedback from the scheduler to the planner enabling the planning heuristic to identify dead-ends earlier and, thus, substantially reducing the search effort.

Our experiments show that our approach leads to a significant reduction in states searched, increased coverage, and improved solve times in a range of International Planning Competition (IPC) benchmark domains. Moreover, because we use a preprocessing detection to identify these semaphore and envelope fact structures, there is no negative impact of our approach on domains where these structures do not exist. This paper demonstrates that CP can be a powerful tool for a kind of combinatorial and temporal reasoning that is traditionally solved inefficiently in the search, which is designed for causal reasoning in AI planners.

These contributions are different from previous applications of Constraint Programming in Temporal Planning – such as CPT [18, 10] or EUROPA [1] – because previous approaches have directly solved the temporal planning problem. Moreover, our approach can model numeric resources and supports a temporal-numeric fragment of PDDL2.1 even though the performance improvements we demonstrate are derived from the temporal aspect of the problem.

2 Background

2.1 Temporal Planning

We consider the fragment of PDDL 2.1 [9] planning problems supported by COLIN and POPF [3] and limit our attention to problems without continuous numeric effects, although our work could be extended to support them.

We define a planning problem as a tuple $\langle F, V, A, I, G\rangle$. F is a finite set of facts. A fact $f \in F$ is a proposition which, if present in a given state s, indicates that the fact is true. V is a finite set of numeric variables. I denotes the initial state. A state is a tuple $\langle F_s, n_s\rangle$, where $F_s \subseteq F$ and n_s is a set of assignments to the variables in V. $n_s[v]$ denotes the value of variable $v \in V$ in state s. A is a set of durative actions, representing operators that can be applied to states. G is a set of conditions that must hold true following execution of any valid solution plan.

We begin by defining a condition ψ. Conditions can either be propositional ($\psi \in F$) or numeric. Numeric conditions are defined as a tuple $\psi = \langle v \ op \ c\rangle$ such that $op \in \{\geq, \leq, <, >$

$, =\}$ and $c \in \mathbb{R}$. For a given state $\langle F_s, n_s \rangle$, ψ is satisfied if it is a proposition where $\psi \in F_s$ or it is a numeric condition where $\langle n_s[v]\ op\ c \rangle$ is true.

A durative action $a \in A$ is defined as a tuple $\langle pre(a)_\vdash, \mathit{eff}(a)_\vdash, pre(a)_\leftrightarrow, pre(a)_\dashv, \mathit{eff}(a)_\dashv, d_a \rangle$. $pre(a)_\vdash$ ($pre(a)_\dashv$) is a set of propositional and numeric conditions (preconditions) on the start (end) of the action a. $pre(a)_\leftrightarrow$ are invariant conditions over the duration of action a that must be satisfied for all the time the action is executing. Individually we denote these preconditions as $pre(a)^p_{\{\vdash,\dashv,\leftrightarrow\}}$ for propositional conditions and $pre(a)^n_{\{\vdash,\dashv,\leftrightarrow\}}$ for numeric conditions. d_a is a pair $\langle d_{min}, d_{max} \rangle$ representing the minimum and maximum duration of the action a as positive real values.

$\mathit{eff}(a)_\vdash$ ($\mathit{eff}(a)_\dashv$) are effects that occur at the start (end) of the action a and consist of a tuple $\langle \mathit{eff}^+, \mathit{eff}^-, \mathit{eff}^n \rangle$. $\mathit{eff}^+(\mathit{eff}^-)$ is a set of propositions in F that are added to (deleted from) a state s to create the subsequent state s' (e.g. $F_{s'} = (F_s \setminus \mathit{eff}^-) \cup \mathit{eff}^+$).

eff^n is a set of numeric effects of the form $\langle v\ op\ c \rangle$, where $op \in \{+=, -=, =\}$, $c \in \mathbb{R}$ and $v \in V$. Numeric effects using the operators $+=$ and $-=$ use the value of v in s, to calculate the value of v in the subsequent state s'. This is a commonly used restricted version of PDDL 2.1 numeric planning.

The goal G is a set of propositional and numeric conditions. A state $s \models G$ if s satisfies the propositional and numeric conditions in G.

A plan π is a list of tuples of the form $\langle a, t, d \rangle$, with t representing the timestamp at which action a is applied and d representing the duration of a where $d_{min} \leq d \leq d_{max}$. A plan π is valid iff for all tuples $\langle a, t, d \rangle$ in π, $pre(a)_\vdash$ is satisfied at time t, $pre(a)_\dashv$ is satisfied at time $t + d$, $pre(a)_\leftrightarrow$ is satisfied at all points in the interval $(t, t + d)$. π is a solution if it is valid and the resulting state S_G, when all actions have finished executing, satisfies G.

2.2 Constraint Satisfaction Problems

Formally, a Constraint Satisfaction Problem (CSP) is a tuple $\langle X, D, C \rangle$, where X is a set of n variables, D is a set of n domains corresponding to the variables in X and $\forall d \in D, d \subset \mathbb{Z}$.

C is a set of *constraints*. A constraint is an m-ary ($m \leq n$) function $c(v_0, ..., v_m) \rightarrow \{true, false\}$ where $v_i \in d_i$ and $d_i \in D$ represents the domain of variable x_i. A constraint can be any mapping of an assignment for the variables in X to a truth value which indicates if the constraint is satisfied or not.

A solution for a CSP $\langle X, D, C \rangle$ is a n-tuple $V = \langle v_0, ..., v_n \rangle$ representing an assignment of the value v_i to the variable $x_i \in X$ where $v_i \in d_i$ and $\forall c \in C, c(V) = true$.

In CP-based scheduling, *interval variables* represent an optional time window. The domain of an interval variable is a set of the form $\{\perp\} \cup \{[s, e) | s, e \in \mathbb{Z}, s \leq e\}$ where s (e) represents the start (end) time of the interval. For an interval variable x, if $x = \perp$ it is not present in the solution to the problem.

A global constraint is a relation on an arbitrary number of variables, typically representing frequently observed combinatorial structure such as a set of variables all requiring pairwise different values. Explicit representation of such relations improves problem solving performance through the use of inference (or propagation) algorithms designed for the corresponding structure [16].

We use three common global constraints: *noOverlap* (or *disjunctive*), *alternative*, and *span*.

▶ **Definition 1.** *noOverlap*(S, υ) *is a global constraint over a set of interval variables* S*, and numeric constant* υ *where for any two present intervals in* S*,* x_i *and* x_j*,* $start(x_i) \geq end(x_j) + \upsilon$ *or* $start(x_j) \geq end(x_i) + \upsilon$ *holds.*

▶ **Definition 2.** *alternative*(x_a, S) *is a global constraint over an interval variable* x_a*, and a set of interval variables* S *where* $x_a \notin S$*.* x_a *is present in the solution, iff exactly one interval* $x \in S$ *is also present. When* x_a *is present,* $start(x_a) = start(x) \wedge end(x_a) = end(x)$*.*

▶ **Definition 3.** *span*(x_c, S) *is a global constraint over an interval variable* x_c *and a set of interval variables* S*,* $x_c \notin S$*, that ensures that all present intervals in* S *are scheduled between the start and end of* x_c *and that interval* x_c *starts with the start of the first present interval in* S *and ends with the end of the last present interval in* S*.* x_c *is absent iff all intervals in* S *are absent.*

2.3 Planning through Decomposition

Decomposition-based planners separate temporal reasoning from causal reasoning by relaxing the problem to a causal representation and finding a solution to this representation using search. In OPTIC, an A^* search with the Metric Temporal Relaxed Planning Graph (TRPG) [13, 6] heuristic is used to explore the state space. To reason about temporal planning, OPTIC splits durative actions into snap actions.

▶ **Definition 4.** *An action* a *of the form* $\langle pre(a)_{\vdash}, eff(a)_{\vdash}, pre(a)_{\leftrightarrow}, pre(a)_{\dashv}, eff(a)_{\dashv}, d_a\rangle$*, can be abstracted as a pair of* ***snap actions*** $\langle a_{\vdash}, a_{\dashv}\rangle$*: a tuple comprising* $\langle pre(a)_{\vdash}, eff(a)_{\vdash}\rangle$ *(*$\langle pre(a)_{\dashv}, eff(a)_{\dashv}\rangle$*). Snap actions represent the instantaneous start and end preconditions and effects of a durative action; the duration and invariant conditions must be tracked separately.*

The search is modified to ensure that each start snap action has a corresponding end action and that invariant conditions between start and end snap actions are satisfied [3].

The successor of a state s reached by applying the sequence of snap actions (partial plan) π_s are each generated by applying a new snap action a, resulting in a new state with partial plan π' (a appended to π_s). From π', OPTIC creates a scheduling problem τ as follows [4]:

- A set of temporal constraints of the form $min \leq t(a') - t(a) \leq max$ where $a, a' \in \pi'$ and min and max are constants.
  - If the temporal constraint represents an ordering between two snap actions $min = \epsilon$ and $max = \infty$. Ordering constraints ensure that (1) if an action has a precondition on a fact p then it is ordered after the last adder of p (i.e., the last action that adds p); (2) if an action adds p it is ordered after the last deletor of p and (3) if an action deletes p it is ordered after the last adder and all conditioners on p since it was last added (4) if an action contditions on or effects a variable v it is ordered after the last action to condition on or effect v.
  - If the temporal constraint represents the duration between the start and corresponding end snap action for duration action a, $min = d_{min}$ and $max = d_{max}$ where $d_a = \langle d_{min}, d_{max}\rangle$. Duration constraints ensure that the time between the start and end snap actions in any valid solution for τ is consistent with duration bounds of the durative action they represent.
- In the presence of continuous or duration-dependent effects these are encoded in the scheduling problem over the values of numeric variables (v in V) before or after each snap action.

In every state generated, a scheduling problem τ is constructed from the partial plan π'. τ is solved using a sub-solver such as an STN-based solver, Linear Program (LP) or Mixed Integer Program (MIP) which attempts to find a set of action timestamps that satisfy the constraints.

If there is no feasible timestamp assignment, the state is temporally inconsistent and is pruned. If an assignment is found and the state satisfies the goal, then the plan π constructed from the timestamps is a solution. If the state is consistent but not a goal, then control returns to the planner to continue its search.

3 Building a CP model from π'

We now construct a drop-in CP replacement for the STN/MIP scheduler currently used in OPTIC, based on the temporal and numeric constraints described in Section 2.3. Consider a partial plan π' and a scheduling problem τ; we construct a CSP $\langle X, D, C\rangle$ as follows.

Variables

For each start snap action $a_\vdash$ in π' we add an interval variable $x_a = [s, e)$ to X regardless of whether $a_\dashv$ exists in π'. s represents the timestamp of snap action $a_\vdash$, and e of $a_\dashv$, if $a_\dashv$ is present. If $a_\dashv$ is not part of the plan π', then we still represent the unclosed duration action that corresponds to $a_\vdash$ with an interval and constrain its duration according to the duration of the durative action it represents, however there will be no ordering constraints on $a_\dashv$ (and consequently on e) because the planner has yet to reason about the addition of $a_\dashv$ to the plan. This allows the scheduler to reason with x_a as if it were a closed action, preserving the duration. In any circumstance, even if the CP solver concludes that a schedule exists for a plan π' with an unclosed action, the plan is not valid and a final temporal consistency check will occur once a complete plan is produced.

If $V \neq \emptyset$ we add two integer variables $x_{a,v}$ and $x'_{a,v}$ for each $v \in V$ and each snap action $a \in \pi'$. These variables represent the value of variable v before and after the application of the snap action.

Temporal Constraints

For each temporal constraint in τ of the form $min \leq t(a') - t(a) \leq max$, a constraint is introduced according to whether the snap action a (a') correspond to a start $a_\vdash$ ($a'_\vdash$) or end $a_\dashv$ ($a'_\dashv$) snap action. Constraints are mapped accordingly as follows. If a and a' are start snap actions then $min \leq start(x'_a) - start(x_a) \leq max$ is introduced. If a and a' are end snap actions then $min \leq end(x'_a) - end(x_a) \leq max$ is introduced. In the case that a is a start snap action and a' is an end snap action (or vice versa) a constraint $min \leq end(x'_a) - start(x_a) \leq max$ ($min \leq end(x'_a) - end(x_a) \leq max$) is introduced.

Numeric Constraints

There are two types of numeric constraints which we model: conditions and effects. A numeric condition as defined in the planning problem with the form $\langle v\{\geq, \leq, <, >, =\}c\rangle$ occurring in $pre(a)$, is imposed on $x_{a,v}$ as it represents the value of the variable v before the application of snap action a e.g. $x_{a,v}\{\geq, \leq, <, >, =\}c$.

If a numeric effect exists in $eff^n(a)$ of the form $\langle v\{+=, -=, =\}c\rangle$ then the resulting constraint is $x'_{a,v} = x_{a,v}\{+=, -=, =\}c$. If no numeric effect exists then $x'_{a,v} = x_{a,v}$.

These constraints model discrete numeric effects within the scheduling problem. In problems where there are no continuous or duration-dependent effects, these need not be reasoned about in the scheduler (since, given the ordering constraints OPTIC generates, the timestamps assigned to actions cannot affect the value of numeric variables), so an STN, without any numeric constraints can be used to solve the scheduling problem. However, in

the presence of either of these, the numeric preconditions and effects must be modelled to ensure temporal-numeric constraints are satisfied. The approach we take to modelling these in our CSP model, mirrors that used in OPTIC. We limit ourselves to discrete effects, here we will use these later in our feedback mechanism from the scheduler to the planner; but in general the same constraints over these variables that are used in OPTIC's MIP can be used to represent continuous/duration dependent effects in a CSP.

In OPTIC, when constructing a scheduling problem τ, a total ordering is imposed between snap actions which condition on or effect the same variable v (regardless of whether this is necessary). We could in principle relax these constraints, but since our focus here is not specifically on numeric planning we maintain them as is. The initial value of $x_{a',v}$, therefore, is constrained to be equal to the final value of v in a ($x_{a',v} = x'_{a,v}$) as we know this would have been the last time the value of the variable v changed. If a is the first snap action π' to condition or effect on v, then $x_{a,v}$ takes the value of v in the initial state ($x_{a,v} = n_I[v]$).

Domain

The domain d_a is defined for an interval variable x_a as the interval $[0, h]$ where h represents the sum of the maximum duration of all actions in the plan π', plus ϵ multiplied by the number of actions in the plan. The horizon h is an upper bound as in the worst case each action will be executed without overlap with any other.

The domain d_v – which applies to all integer variables representing v – is an interval $[min_v, max_v]$ defined by the sum of all positive numeric effects ($+=$) effects applied to the initial value max_v, and the sum of all negative effects ($-=$) min_v. In the event assignment ($= c$ where c is some constant) occurs, the interval is $[c - min_v, c + max_v]$.

Temporal planning and CP scheduling problems have adopted different conventions for representing the timing of events. As shown in the definition of an interval variable, in CP, time intervals are considered to be open on the right, thus allowing one interval to end at time t and another to begin at the same time even if they are constrained to not overlap. In contrast, temporal planning uses ϵ as the smallest representable unit of time. Actions that are constrained to not overlap must be separated by at least ϵ. To handle this mismatch, we represent ϵ in the CP model as one unit of time and thus force an extra gap between actions, consistent with the planning definition. Our CP scheduler does not affect OPTIC's support for self-overlapping actions, is equivalent to the current STN-based approach in OPTIC, and as such does not impact the soundness or completeness to OPTIC.

4 Abstracting Semaphores and Envelopes

So far our temporal reasoning problem is identical to that solved by the STN solver in OPTIC. In this section, we extend our temporal representation to take advantage of CP's greater expressivity and solving power.

Semaphore and *envelope* facts [11] are causal modelling patterns that, respectively, prevent and require the concurrent execution of a set of actions. Because these are facts, and therefore a causal consideration, they have been considered during causal reasoning, despite being a temporal structure. We formally define *semaphore* and *envelope* facts as follows.

▶ **Definition 5.** *A* ***semaphore fact*** f *is a fact such that* $f \in I$ *and* $\forall a \in A$ *exactly one of the following holds*

- a *is a* ***mutual exclusive action****, that is,* f *is in and only in* $pre(a)^p_\vdash$, $\mathit{eff}(a)^-_\vdash$ *and* $\mathit{eff}(a)^+_\dashv$,
  or

- *a is an unrelated action, that is f is not in any precondition or effect of a.*

Definition 5 ensures that all actions that condition on a *semaphore fact* delete it at the start and add it at the end, ensuring actions that condition on a semaphore are mutually exclusive (cannot execute in parallel). Requiring all other actions to be unrelated ensures the semaphore fact serves only to enforce mutual exclusion and has no other function.

▶ **Definition 6.** *An **envelope fact** f is fact such that $f \notin I$ and $\forall a \in A$ exactly one of the following holds*

- *a is an **envelope achiever**, that is, f is only in $\mathit{eff}(a)^{+}_{\vdash}$ and $\mathit{eff}(a)^{-}_{\dashv}$,*
- *a is an **envelope conditioner**, that is, f is in $\mathit{pre}(a)^{p}_{\leftrightarrow}$ and optionally in $\mathit{pre}(a)^{p}_{\vdash}$ and/or $\mathit{pre}(a)^{p}_{\dashv}$ and f is not in $\mathit{eff}(a)_{\vdash}$ and $\mathit{eff}(a)_{\dashv}$, or*
- *a is an unrelated action, that is, f is not in any of the sets: $\mathit{pre}(a)^{p}_{\vdash}$, $\mathit{eff}(a)_{\vdash}$, $\mathit{pre}(a)^{p}_{\leftrightarrow}$, $\mathit{pre}(a)^{p}_{\dashv}$, or $\mathit{eff}(a)_{\dashv}$.*

An *envelope fact* ensures that every envelope conditioner executes concurrently with some envelope achiever. Definition 6 ensures that an *envelope achiever* adds the envelope fact at its start and deletes the same fact at its end, thus creating a window where this fact is available. The definition also states that an *envelope conditioner* has an invariant condition and thus must execute concurrently with some envelope achiever. An unrelated action as defined in Definition 6 means an action can only act as an achiever or conditioner on an envelope fact and not use an envelope fact for any other purpose.

Definition 6's requirement for all other actions to be unrelated ensures that (i) no other actions adds the envelope fact, so all conditioners *must* occur within an achiever, (ii) no other actions delete the envelope fact, thus it remains throughout the entire execution of an envelope achiever, and (iii) no other actions condition on the envelope facts at only the start or end and thus only need to be executed partially concurrently with an achiever.

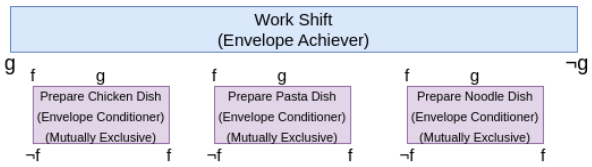

Figure 1 Envelope fact g enforces concurrency with a *work shift* whilst semaphore fact f prevents more than one *work activity* from happening at a time. A fact appearing above (below) an action indicates it is a condition (effect) respectively. Position indicates whether the fact is a start, invariant, or end condition or effect. $\neg f$ denotes that f is being deleted (made false).

Figure 1 shows how a semaphore f and envelope g interact. The blue "work shift" action is an envelope achiever of g, into which a number of work activities have to be scheduled. These activities – preparing Chicken, Pasta and Noodle dishes – are mutually exclusive due to semaphore fact f and are also envelope conditioners on g.

4.1 Abstracting Semaphore Facts

Current state-of-the-art decomposition planners search over all total ordering constraints between durative actions a_i and a_j that condition on semaphore fact f.

Figure 2 shows how the mutual exclusion of semaphore f creates two alternate plans to explore to achieve the same state. If there are n actions required to achieve goal G that condition on the semaphore fact f, there are $n!$ orderings of those actions to be considered.

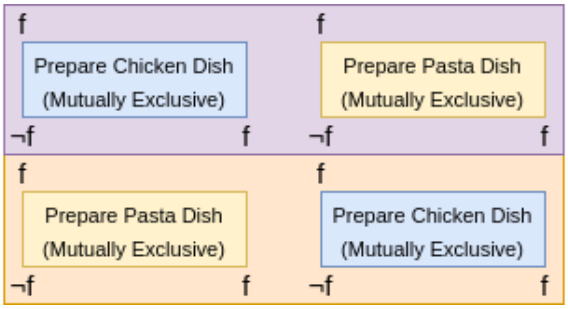

Figure 2 For two mutually exclusive actions, two orderings exist that achieve the same state.

However in many cases, side effect constraints and other optimisations [5] can reduce the search space.

To abstract a semaphore fact f, from a planning problem $\langle F, V, A, I, G\rangle$, we create a set of actions $m_f = \{a : a \in A, f \in pre(a)^p_\vdash\}$. We perform the following set operations $\forall a \in m_f$: $pre(a)^p_\vdash \setminus \{f\}$, $\mathit{eff}(a)^-_\vdash \setminus \{f\}$, and $\mathit{eff}(a)^+_\dashv \setminus \{f\}$. Finally, we remove f from the initial state I.

For the scheduling problem corresponding to partial plan π', a new *mutual exclusion constraint* is added for the actions in π' that appear in the set of actions m_f.

▶ **Definition 7.** *A mutual exclusion constraint is a noOverlap constraint of the form* $\textbf{\textit{noOverlap}}(\{x_a \in X : a \in \pi' \wedge a \in m_f\}, \epsilon)$.

Following Definition 5, we demonstrated that the only purpose of a semaphore fact was to ensure mutual exclusion between actions conditioning on it. A semaphore fact has no implied ordering between actions that condition on it. Search in OPTIC imposes a total ordering among such actions (due to ordering each conditioner after the most recent adder), with different orderings considered by exploring different plans. We replace the total orderings imposed at the planning level, with a mutual exclusion constraint imposed during scheduling. The scheduler will consider any ordering of actions in the mutual exclusion constraint that respects ordering constraints imposed by other facts (since these still remain as temporal constraints). As a result, the substitution preserves soundness. By considering all sound partial orderings, the substitution maintains completeness.

4.2 Abstracting Envelope Facts

An envelope achiever of fact g creates a time window that an envelope conditioner on g must execute within. With multiple achiever and conditioner actions, a set of possible time windows is defined and the planner must decide which envelopes each conditioner must execute within.

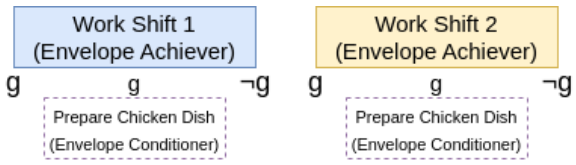

Figure 3 For two envelope achievers (work shift 1 and 2); prepare chicken could be assigned to either.

In Figure 3 two achievers of g – "Work Shift 1" and "Work Shift 2" – exist and "Prepare Chicken Dish" could be scheduled within either; resulting in two different assignments to

consider. The number of assignments grows exponentially with both the number of envelope achievers n and the number of envelope conditioners m provided $n > 1$.

Assignment decisions are only important if envelope conditioners cannot be executed concurrently. If all envelope conditioners can execute concurrently, an envelope achiever large enough to execute concurrently with the longest conditioner can satisfy all conditioners.

To abstract an envelope fact g from a planning problem $\langle F, V, A, I, G\rangle$, we create two sets of actions $achievers_g = \{a : a \in A, g \in \mathit{eff}(a)^+_\vdash\}$ and $conditioners_g = \{a : a \in A, g \in pre(a)^p_\leftrightarrow\}$. $\forall a \in achievers_g$, we perform $\mathit{eff}(a)^-_\dashv \setminus \{g\}$. $\forall a \in conditioners_g$ we perform the following set operations: $pre(a)^p_\leftrightarrow \setminus \{g\}$, $pre(a)^p_\dashv \setminus \{g\}$, and $pre(a)^p_\vdash \cup \{g\}$.

To ensure that envelope conditioners are scheduled concurrently with an envelope achiever, a new *envelope constraint* comprised of new variables and a set of constraints is added to the scheduling problem for the partial plan π'.

▶ **Definition 8.** *An envelope constraint for an envelope fact g, with sets of actions $achievers_g$ and $conditioners_g$ and partial plan π', comprises new interval variables $x_{a,c}, \forall a \in achievers_g, \forall c \in conditioners_g$, and a dummy optional interval $x_{a,dur}$ for all $a \in achievers_g$ and the following constraints:*

- *$\forall c \in \pi$ that also appears in $conditioners_g$, $alternative(x_c, \{x_{a,c} : a \in achievers_g \cap \pi'\})$. x_c is the interval variable in X representing the conditioner action c. Exactly one achiever, a, for each conditioner is assigned by enforcing the presence of one optional interval variable $x_{a,c}$.*
- *$\forall a \in \pi'$ that also appears in $achievers_g$, $span(x_a, \{x_{a,c} : c \in conditioners_g\} \cup \{x_{a,dur}\})$. x_a is the interval variable in X representing the achiever action a. This ensures that each envelope conditioner executes concurrently with the envelope achiever assigned in the alternative constraint.*

The dummy optional interval $x_{a,dur}$ ensures that the conditioners on an envelope do not have to be scheduled such that one starts exactly at the start of x_a and one finishes exactly at the end. The dummy action can be used to satisfy this condition imposed by the span constraint, and thus we do not compromise completeness (as the planning model does not necessarily imply this constraint).

Following Definition 6, we demonstrated that the only purpose of an envelope fact in the domain was to ensure that all envelope conditioners execute entirely within envelope achievers. An envelope fact does not imply an assignment of a specific conditioner to a specific achiever.

In OPTIC, an assignment of a conditioner to an achiever is done during search. Search first adds the start of an achiever ($a_\vdash$), then the start of the conditioner ($c_\vdash$), imposing an ordering constraint in τ: $t(a_\vdash) < t(c_\vdash)$. Search then adds the end of the conditioner ($c_\dashv$). Finally, search adds the end of the achiever ($a_\dashv$), ordering the end of the achiever after the end of the conditioner ($t(c_\dashv) < t(a_\dashv)$). The result of these constraints is a total ordering $(t(a_\vdash) < t(c_\vdash) < t(c_\dashv) < t(a_\dashv))$.[1]

In imposing these ordering constraints in the scheduling problem τ, search assigns a conditioner to an achiever and enforces their concurrent execution using the same constraints. Search in OPTIC considers a different assignment of an achiever to a conditioner, by performing the same process described above, but using different achievers.

[1] OPTIC may choose to add some other action or actions between the addition of these individual snap actions. This is a simple example of how envelope concurrency is achieved.

By abstracting an envelope fact, leaving only a start add effect for achievers and a start precondition for conditioners, the only ordering constraint added by OPTIC to τ orders all conditioners after the first achiever; an ordering implied in any valid envelope assignment and execution. By using an *alternative* constraint, the scheduler can consider the assignment of a conditioner to any achiever. Meanwhile, the *span* constraint ensures the concurrent execution of a conditioner with the achiever it is assigned to.

Because we only abstract envelope facts, other constraints in τ are preserved (still generated by OPTIC's machinery). The assignment of a conditioner to an achiever will only be considered by the CP Scheduler if that assignment respects all other constraints in τ. As a result, this abstraction does not compromise soundness.

By allowing the CP Scheduler to consider different assignments of conditioners to achievers, we ensure that, in one state, the scheduler considers all orderings that OPTIC considers over a number of states. We therefore do not compromise completeness.

5 Improving Feedback to the Planner

The abstractions described in Section 4 and allow us to transfer some reasoning about orderings of actions in the planner's search to reasoning with powerful global constraints in a CP sub-solver. These abstractions also allow us to enhance the communication between the causal reasoning in task planning and the temporal reasoning in scheduling.

In delete-free relaxation heuristics [12], envelopes are a particular challenge [11] because once an envelope fact g is achieved, there is no further value to the heuristic to achieve g again. Without reasoning about how much time is required for envelope conditioners, search has to blindly add achievers until sufficient time is available for the scheduler. Yet if envelopes have limited time, it is critical that further achievers are added. The need for multiple achievers is greatest when conditioners are mutually exclusive

In Section 4.2, when an envelope fact g was abstracted, the add effects in all achievers were preserved and a start precondition was added to all conditioners on g. This transformation forces search to add at least one achiever prior to adding any conditioners for envelope fact g.

To guide search, we now add a new subset of numeric variables to the set V for a planning problem $\langle F, V, A, I, G\rangle$. These new numeric variables – *Envelope Time Tracking variables* – create a producer-consumer relationship between envelope achievers and mutually exclusive conditioners. Envelope Time Tracking variables are added for each semaphore fact f and envelope fact g.

▶ **Definition 9.** *A time tracking variable is a numeric variable $v_{f,g} \in V$. The initial state value of variable $v_{f,g}$ is* 0 *($n_I[v_{f,g}] = 0$).*

Each achiever $a \in achievers_g$ has a new start effect $v_{f,g}$ += d_{max}. Meanwhile each conditioner $c \in conditioners_g \cap m_f$ has a new start condition $v_{f,g} \geq d_{min}$ and a new start effect $v_{f,g}$ −= d_{min}.

The variable $v_{f,g}$ increases by the maximum duration of each achiever of g added to π'. The variable $v_{f,g}$ decreases by the minimum duration of each conditioner of g in m_f added to the partial plan π'. $v_{f,g}$ represents a trivial upper bound on the amount of free time remaining in envelope achievers for envelope conditioners in mutual exclusion m_f.

The introduction of the precondition $v_{f,g} \geq d_{min}$ means that, across all achievers, there must exist enough free time to fully contain a conditioner before the search can add it. Pruning based on the value of $v_{f,g}$ does not compromise completeness because it is an overestimate and so we necessarily need to add another achiever before we can fit any further conditioners.

5.1 Prioritising More Easily Schedulable States

If π' is shown to be temporally consistent, the scheduler produces a plan π which includes a valid schedule for all actions in the plan. We can use this schedule to identify when adding a new action to the plan will likely lead to a trivially schedulable partial plan; and when we might need to add another envelope achiever in order to find a solution.

For a given envelope achiever a in π we can compute the maximum free time, $v^a_{f,g}$, by summing the available free time in a in the computed schedule.

▶ **Definition 10.** *For an envelope achiever a, $v^a_{f,g} = d_a - \sum d_c$, $\forall c \in conditioners_g \cap m_f$ where conditioner c is in the plan π and scheduled concurrently with a (i.e. in π, t_c is in the interval $[t_a, t_a + d_a]$).*

Variable $v^a_{f,g}$ allows us to determine the maximum free space within a single envelope achiever a in a plan π. Figure 4 shows an example of the calculation in Definition 10, where the maximum possible free space within the envelope a is 10 time units.

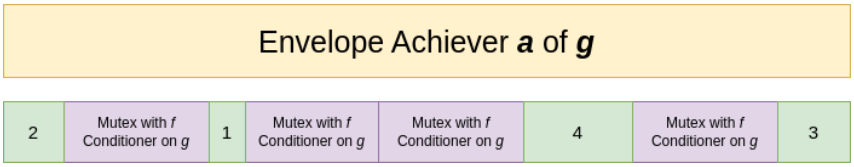

Figure 4 illustrates an envelope achiever a of envelope fact g to which several conditioners have been assigned in a plan.

$v^\pi_{f,g}$ is calculated for a plan π using Definition 10, a semaphore fact f and envelope fact g as follows:

▶ **Definition 11.** *For a plan π, a semaphore fact f, and an envelope fact g is $v^\pi_{f,g} = max(v^a_{f,g})$ (Definition 10), $\forall a \in achievers_g$, where a is in the plan π.*

$v^\pi_{f,g}$ gives us an estimate across the plan π of the longest conditioner we could schedule within any achiever. Whilst this bound is a good estimate of the upper bound, it is not a guaranteed maximum because it is possible that reassigning conditioners to different achievers could increase the value of $v^a_{f,g}$ for some achiever a. Thus, we cannot use $v^\pi_{f,g}$ for pruning as we do with $v_{f,g}$. Instead we favour expanding states that are more likely to be schedulable: any state generated by applying an action for which $v^\pi_{f,g} \leq d_{min}$ is added to a second open list that is only expanded if the first open list becomes empty.

This manipulation guides search to favour adding conditioners whose duration fits within an existing achiever or adding another achiever first; before attempting to add further conditioners. Such a preference is useful in counteracting the heuristic blind spot discussed earlier, in which there is no heuristic guidance to open new envelopes. Prioritisation of states in this way does not compromise completeness because, whilst it would not be sound to prune states based on $v^\pi_{f,g}$, the use of a second open list ensures that these states will be explored eventually if required.

This type of communication between the planner and scheduler is novel. Where previous communication has been limited to constraints and inconsistencies, here the scheduler is communicating temporal information that is incorporated into the planner's search, opening the possibility of the communication of other temporal search guidance.

IPC Domain	No. of Problems	Type
cafe	29	both
crew planning	29	both
driverlog (shift)	20	envelopes only
match	20	both
pipes (no tankage)	30	envelopes only
turn and open	20	envelopes only
satellites	30	both
TMS	20	both

Table 1 Benchmark domains used.

6 Evaluation

To evaluate our transformations, we identify a wide variety of International Planning Competition (IPC) domains from across a number of years that contain envelopes and (optionally) semaphores. Table 1 shows the domains used. The use of our transformations and the CP scheduler is automatically decided based on the presence of envelope and/or semaphore facts, in a preprocessing step that takes $< 0.001s$, if none are detected the planner runs exactly as before. Therefore performance on domains which do not contain envelopes or semaphore facts is unaffected; thus we do not consider these in our evaluation.

We compare the performance of the CP Scheduling approach to the baseline standard approach in OPTIC. The two configurations we tested are:

- The *base configuration* is a version of OPTIC [2], a leading general-purpose temporal-numeric planner. OPTIC uses an STN-based scheduler with a P-time complexity.[2]
- The *experimental configuration* is the base configuration plus a CP scheduler that includes our abstraction of semaphores and envelopes and the time tracking variables. The CP scheduler is configured to return the first feasible solution found. The search and memoization machinery of OPTIC is otherwise unchanged.

We experimented with a configuration that caches the scheduler solution to allow warm starting in subsequent states. This was not successful because the memory overheads of storing a CP solution in every state are too high, and the planner began to hit the memory limit.

All problems were executed on an Intel i7-8650U 1.9GHz machine with 3GB of memory and a search time limit of 30 minutes. We use IBM's CP Optimizer 22.1.0 as our CP sub-solver. Each plan is validated using VAL [14]. We note that all unsolved problems were the result of search timeout, rather than memory limits.

The objective of this evaluation is to investigate whether the transformations reduce the number of states explored, by replacing multiple planning states with a single state with a less constrained partial order plan and increased search guidance through temporal feedback from the scheduler. We go on to investigate whether planner performance is improved as a result. Thus our evaluation focuses on comparing the two scheduling approaches within the same planner.

[2] OPTIC has both an STN and a MIP scheduler which it chooses based on the nature of the problem. None of these problems have continuous numerics, so OPTIC defaults to the STN scheduler.

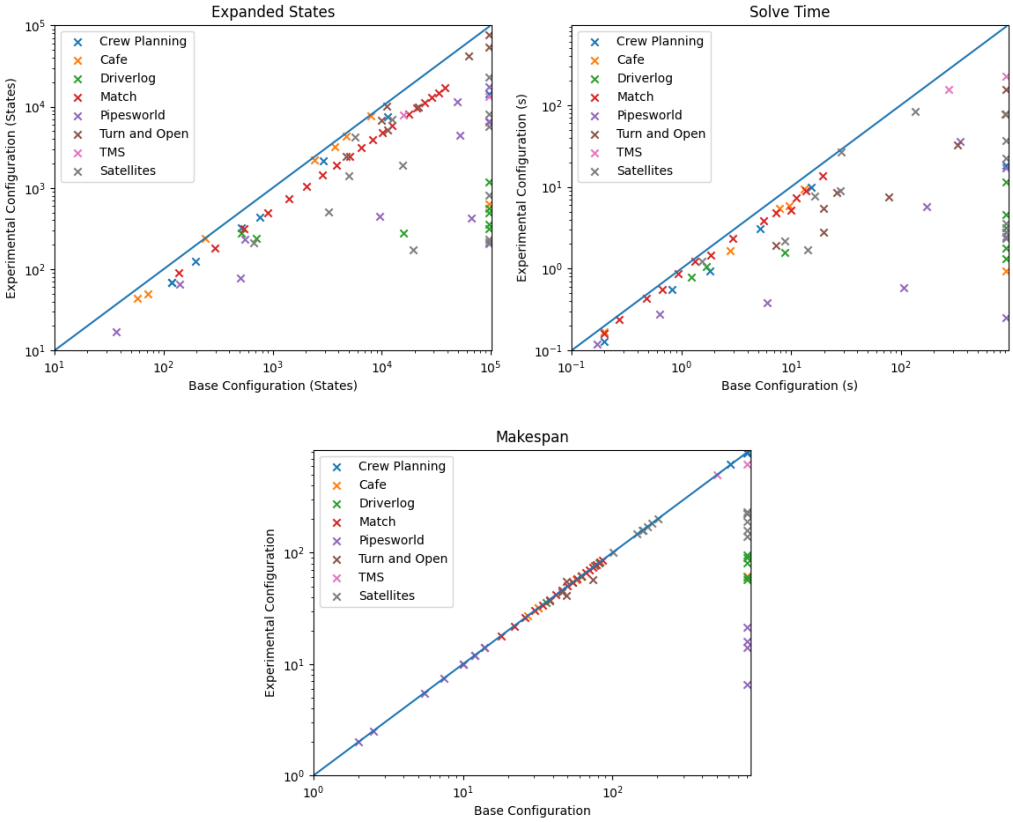

Figure 5 Comparison of base and experimental configuration, by states, time and makespan. Points on the far right/top axis indicate problems not solved by the base/experimental configuration respectively.

6.1 Reduction in States Generated

The graph of *states generated* in Figure 5, shows that the experimental configuration reduces the number of states generated across all domains. The shift of the disjunctive reasoning, created by semaphore and envelope facts, from the planner to the scheduler means fewer disjunctive decisions are made in planning search and consequently fewer states are generated.

Domains with a modest reductions in the number of states generated, such as the Match domain and Cafe domain, are a result of the underlying envelope and semaphore structure. In the Match domain, the goal is to mend a set of fuses in a power outage where matches must be lit to provide light. Each mend fuse action is mutually exclusive, and any fuse can be fixed whilst any match is lit. Each match can be used to repair at most two fuses, and the goal is for all fuses to be fixed with the matches provided. Matches are symmetrical; therefore, reordering how matches are lit does not make a temporally inconsistent partial plan consistent, so abstracting mutual exclusion is redundant. For the envelope assignment problem that exists between matches and fuses, each match is equally capable of mending each fuse. Thus the choice of assignment is also redundant. The only benefit of the transformations described is in guiding search to add sufficient matches to mend all fuses.

Cafe has a similar structure to Match, with three distinctions. Firstly, Match consists of one “temporal” resource being consumed: matches. In Cafe, there are two symmetrical

IPC Domain	Problems	Base	Experimental
cafe	29	7	8
crew planning	29	7	8
driverlog	20	3	8
match	20	19	19
pipes	30	8	12
turn and open	20	8	8
satellites	30	7	13
TMS	20	1	2

Table 2 No. of Problems solved by base and experimental configurations.

resources, Ovens and Cooks. Ovens and Cooks are not a consumable resource, i.e. they can be reused to complete other actions. The final distinction is that the meals being prepared have different durations, whereas each fuse takes the same amount of time to fix in Matches. Despite these differences, they are otherwise very similar domains and therefore experience a similar issue where searching symmetrical space yields similar generated state space to the base configuration.

6.2 Coverage

Table 2 shows the coverage (number of problems solved) for the base and experimental configurations. The experimental configuration performs as well or better than the baseline for all domains, solving a superset of the instances that the baseline solves. In the Driverlog, Satellites and Pipes domains the experimental configuration increased coverage dramatically. Across the 198 problems we evaluated, coverage rose from 30.3% in the base configuration to 39.3% in the experimental configuration. Because these transformations are only applied in domains where semaphores or envelopes exist, coverage in other domains is unaffected.

The improvements in coverage are a reflection of the reduction in states generated. The Driverlog and Pipes domain, which experienced larger reductions on smaller problems, increased coverage as their respective problem sets scaled.

6.3 Search Time

The graphs of *states generated* and *search time* in Figure 5 have a similar spread, with a downward shift for points in the search time graph. To understand this shift, Table 3 presents the states generated per second. The experimental condition has an average 1.69 fold increase in time taken per state compared to the baseline.

The increased time per state is a result of the two types of scheduler used. The base configuration solves a P-time scheduling problem each time a state is generated whereas the experimental configuration solves an NP-complete scheduling problem. As a result of the difference in complexity, it is to be expected that the CP solver takes longer per state. A sufficiently large reduction in the number of states generated results in a reduction in the search time, in spite of the increased overhead per state, as seen by the solutions that timed-out for the *base configuration*.

In domains where the experimental configuration outperforms the base configuration in states per second (Cafe, Crew Planning and Turn and Open), we attribute this to the heavily constrained nature of the scheduling problems within these problems. This constraining makes the search space for the CP solver significant smaller when compared to other problems.

IPC Domain	Base	Experimental	Ratio
Cafe	882	992	0.89
Crew Planning	747	759	0.98
Driverlog	891	204	4.37
Match	8696	5703	1.52
Pipes	1339	1224	1.09
Turn and Open	354	1184	0.3
Satellites	475	178	2.67
Average	**1912**	**1464**	**1.69**

Table 3 States per second, and ratio (base/experimental). TMS excluded due to insufficient data.

6.4 IPC Benchmark Score

In the temporal track of the IPC, a benchmark score is used to compare planners [17]. For a given problem, let T^* be the minimum search time in seconds required by any planner to solve the problem. A planner that solves the problem in search time T (in seconds) gets a score of $\frac{1}{1+log_{10}(T/T^*)}$. If a configuration does not solve a problem, it receives a score of 0. Search times of less than one second are rounded up.

Across the domains presented in this evaluation, the base configuration achieved a score of 41.6, whilst the experimental configuration achieved one of 78.

To restate, these transformations and the CP scheduler are only applied in problems where a semaphore or envelope fact is detected. Thus, these transformations represent a net improvement on the IPC score of the base planner. The performance of OPTIC in all other domains remains unaffected.

6.5 Makespan

Makespan is broadly equivalent across all mutually solved problems. This is primarily because the envelope actions somewhat artificially determine the makespan of the plan, even if the conditioner actions are scheduled more efficiently, so the scope for improving makespan is limited in most of domains. The one exception is a slight reduction in makespan for the experimental configuration in the Turn and Open domain. The difference can be attributed to the reallocation of disjunctive reasoning to the CP scheduler.

States generated during search by the base configuration are more constrained and have a stricter ordering than propositionally and numerically equivalent states generated by the experimental configuration. A single state represents more partial-orderings of the relaxed plan that achieves it in the experimental configuration. This can result in a reduction or increase of the makespan depending on the solution that the CP scheduler finds to the envelope allocation and mutual exclusion problems.

6.6 Evaluating Individual Abstractions

As part of our evaluation, we also evaluated abstractions individually. We saw no significant performance improvements from abstracting envelopes or semaphores alone: our improvements lean heavily on the interactions between envelopes and semaphores, in particular, the constraining effect of envelopes relative to the consumptive effect of semaphores.

We also evaluated using the envelope and semaphore abstractions without the time tracking variables defined in Section 5. Experimentally, we saw that the number of expanded

states is 93% of that of the base configuration compared to the experimental configuration's 48%. The overall run-time was similarly impacted. When excluding Time Tracking Variables, there was no increase in coverage or improvement in makespan compared to the base configuration. These results suggest that the enhanced search guidance provided by the novel communication from the sub-solver embodied in these variables is a key to the improved performance that we observed.

7 Conclusion

By abstracting semaphores and envelopes, we remove two forms of disjunctive temporal reasoning from the planning level that cause exponential state space growth in temporal-numeric planning problems. Introducing new numeric variables to represent the available time within envelope achievers and using these variables as a means to communicate remaining envelope time to the planner further guides search and further reduces the state space.

The reductions in search space show that there are significant improvements to be made in more complex decomposition approaches. This new abstraction and decomposition demonstrates that a more expressive scheduler can improve coverage, reduce search space and reduce search times. This work opens the door to further exploration of how planning decisions are divided between task planning and CP-based scheduling and how an appropriate division, further exploiting the strengths of the CP sub-solver, can be used to yield improvements in coverage and speed.

References

1 Javier Barreiro, Matthew Boyce, Minh Binh Do, Jeremy D. Frank, Michael Iatauro, Tatiana Kichkaylo, Paul Henry Morris, James C. Ong, Emilio Remolina, Tristan B. Smith, and David E. Smith. Europa : A platform for ai planning, scheduling, constraint programming, and optimization. 2012. URL: https://api.semanticscholar.org/CorpusID:208653087.

2 J. Benton, Amanda Jane Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In Lee McCluskey, Brian Charles Williams, José Reinaldo Silva, and Blai Bonet, editors, *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, ICAPS'12, pages 2–10. AAAI, 2012. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4699.

3 Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Temporal planning in domains with linear processes. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1671–1676, 01 2009. URL: http://ijcai.org/Proceedings/09/Papers/279.pdf.

4 Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, pages 42–49. AAAI, 01 2010. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1421.

5 Amanda Jane Coles and Andrew Ian Coles. Have I been here before? state memoization in temporal planning. In Amanda Jane Coles, Andrew Coles, Stefan Edelkamp, Daniele Magazzeni, and Scott Sanner, editors, *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, pages 97–105. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13187.

609 **6** Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. Planning with problems requiring
610 temporal coordination. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-*
611 *Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July*
612 *13-17, 2008*, pages 892–897. AAAI Press, 01 2008. URL: http://www.aaai.org/Library/
613 AAAI/2008/aaai08-142.php.
614 **7** Minh Binh Do and Subbarao Kambhampati. Sapa: A multi-objective metric temporal planner.
615 *J. Artif. Intell. Res.*, 20:155–194, 2003. doi:10.1613/jair.1156.
616 **8** Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive
617 heuristic for temporal and numeric planning. In Alfonso Gerevini, Adele E. Howe, Amedeo
618 Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on*
619 *Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.
620 AAAI, 01 2009. URL: http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/742,
621 doi:10.1007/978-3-642-25116-0_6.
622 **9** Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning
623 domains. *J. Artif. Intell. Res.*, 20:61–124, 2003. arXiv:1106.4561, doi:10.1613/jair.1129.
624 **10** Antonio Garrido, Marlene Arangú, and Eva Onaindia. A constraint programming formulation
625 for planning: from plan scheduling to plan generation. *J. Sched.*, 12(3):227–256, 06 2009.
626 doi:10.1007/s10951-008-0083-7.
627 **11** Keith Halsey, Derek Long, and Maria Fox. CRIKEY - a temporal planner looking at the
628 integration of scheduling and planning. In *Proceedings of the Workshop on Integration*
629 *Scheduling Into Planning at Thirteenth International Conference on Automated Planning and*
630 *Scheduling*, 01 2003.
631 **12** Jörg Hoffmann. FF: the fast-forward planning system. *AI Mag.*, 22(3):57–62, 09 2001.
632 doi:10.1609/aimag.v22i3.1572.
633 **13** Jörg Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric
634 state variables. *J. Artif. Intell. Res.*, 20:291–341, 2003. doi:10.1613/jair.1144.
635 **14** Richard Howey and Derek Long. Val's progress: The automatic validation tool for PDDL2.1
636 used in the International Planning Competition. In *Proceedings of the ICAPS 2003 workshop*
637 *on "The Competition: Impact, Organization, Evaluation, Benchmarks"*, 11 2003.
638 **15** Wen-Yang Ku and J Christopher Beck. Revisiting off-the-shelf mixed integer programming and
639 constraint programming models for job shop scheduling. *Computers & Operations Research*,
640 73:165–173, 2016.
641 **16** Jean-Charles Régin. Global constraints: A survey. In *Hybrid Optimization: The Ten Years of*
642 *CPAIOR*, pages 63–134, New York, 2011. Springer.
643 **17** Mauro Vallati, Lukás Chrpa, Marek Grzes, Thomas Leo McCluskey, Mark Roberts, and
644 Scott Sanner. The 2014 international planning competition: Progress and trends. *AI Mag.*,
645 36(3):90–98, Sep. 2015. doi:10.1609/aimag.v36i3.2571.
646 **18** Vincent Vidal and Hector Geffner. Branching and pruning: An optimal temporal POCL
647 planner based on constraint programming. *Artif. Intell.*, 170(3):298–335, 03 2006. doi:
648 10.1016/j.artint.2005.08.004.