When Quality Matters: Constraint Programming for Automated Temporal and Numeric Planning

Roland Godet
LAAS-CNRS, INSA and DTIS, ONERA
Université de Toulouse, France
rgodet@laas.fr

Arthur Bit-Monnot
LAAS-CNRS, CNRS, INSA
Université de Toulouse, France
abitmonnot@laas.fr

Charles Lesire-Cabaniols
DTIS, ONERA
Université de Toulouse, France
charles.lesire@onera.fr

***Abstract*—Automated planning is a field of Artificial Intelligence interested in finding a set of actions that drives the evolution of the environment from an initial state to a desired goal state. Much of the work of the community has been on so-called domain-independent planning where a solver is expected to produce a plan from an abstract problem, specified in a common description language. This has led the community to produce a number of highly-efficient solvers that can be expected to work on a large variety of domains without any fine-tuning.**

Where most of the work has focused on sequential plans over purely symbolic states, we instead propose a constraint-based planner whose focus is on more expressive variants, namely numeric and temporal planning, essential in many practical applications. We extend an existing CP encoding of temporal planning with support for optimization and numeric states and leverage an existing lazy clause generation CP solver to find and optimize plans. Where the most successful automated planners rely on some form of forward-search, we show that constraint-programming can be just as effective in finding satisfiable solutions while substantially improving the quality of the produced plans.

I. Introduction & Related Work

Time and resources have played a strong role in the early days of planning technologies. Several approaches historically leveraged constraint programming solvers to reason jointly on the causal, temporal and metric relationship underlying candidates plans [1]–[4]. The introduction of heuristic state-space search however shifted the focus of the community towards an explicit forward exploration of the reachable states, guided by very elaborate heuristics that steer a best-first search algorithm towards the desired goal state [5], [6].

With the definition of PDDL [7] and the International Planning Competition (IPC), state-space search became the de-facto standard for domain-independent automated planning. The introduction of time and numerics in PDDL2.1 [8] did not change this, with the most successful planners extending the key components of state-space search for concurrent actions [9]–[13].

In parallel, a number of compilation-based planners have been introduced for classical [14], [15], temporal [16], [17] and numeric [18]–[20] planning, that have proved efficient in the setting where only provably optimal plans are allowed. Like forward heuristic search planners, those planners rely on a timed-indexed encoding, resulting in an explicit representation of all states traversed by the plan.

On the other hand, the plan-space and timeline approaches from IxTeT [1] or Europa [21], closer in spirit to scheduling solvers and that leveraged constraint programming, have been phased out due to their inability to match the scalability of state-space search in a domain-independent setting. A notable exception is the work on lifted plan-space planning of FAPE [22] and LCP [23], without however support for numeric states or optimization.

The landscape of numeric planning has evolved to encompass different levels of expressiveness. Simple Numeric Planning (SNP) [24] restricts numeric variables to increase and decrease operations with constants, while Linear Numeric Planning (LNP) [25], [26] extends SNP with direct assignment and linear combinations in effect expressions.

In this paper, we *adapt the pure-temporal SMT encoding* of LCP [23], based on chronicles, to an action-based formalism compatible with PDDL2.2 [27]. We *extend the encoding with full support for LNP features and optimization*. This results in an expressive temporal and numeric planner that differs by its *fully lifted-nature* and its exploitation of a *plan-space* search space whose exploration is delegated to a constraint programming solver. The resulting planner is competitive on established temporal and numeric PDDL benchmarks with state-of-the-art state-space planners. Finally, a warm start is used to further improve performance, out-performing state-of-the-art state-space planners in solution quality.

II. Lifted Action-Based Formalism

In this section, we introduce our lifted action-based formalism used to represent a numeric and temporal planning problem, for the largest part compatible with PDDL2.2 [27].

The *time representation* is assumed to be discrete with a fixed granularity. It is defined from the *temporal origin* (0) to an *horizon* (H) by the set of rationals $\mathcal{T} = \{ 0, \varepsilon, 2\varepsilon, \ldots, H \}$, where $\varepsilon \in \mathbb{Q}^+$ is the *temporal discretization step*.[1]

The *objects* of the problem's environment (*e.g.*, the robots or locations) are represented by a set of symbols $\mathcal{O}$.

As the proposed formalism is lifted, many constructs use *variable parameters*. Each variable parameter is a symbolic variable with a domain $\mathcal{O}' \subseteq \mathcal{O}$.

Part of this work has been supported by the HumFleet project ANR-23-CE33-0003 and has benefitted from the AI Interdisciplinary Institute ANITI. ANITI is funded by the France 2030 program under the Grant agreement n°ANR-23-IACL-0002.

[1] In our implementation, H is set to $2^{31} \times \varepsilon$.

Definition 1 (State Variable)**.** A *state variable* $sv(x)$ represents a dynamic property of the environment, defined by its name sv and a list of variable parameters $x = [\, x_1, \ldots, x_n \,]$. Its value at time $t \in \mathcal{T}$ is denoted as $[t]sv(x)$. There are three types of state variables:

- A *boolean state variable* is a proposition that can be either true ($\top$), false ($\bot$) or undefined ($\dagger$). The set of boolean state variables is denoted as $\mathcal{SV}_{\mathcal{B}}$.
- A *symbolic state variable* can take any value within a given set of symbols $\mathcal{O}' \subseteq \mathcal{O}$ or be undefined ($\dagger$). The set of symbolic state variables is denoted as $\mathcal{SV}_{\mathcal{S}}$.
- A *numeric state variable* can take any integer value within a given range $[\![lb(sv), ub(sv)]\!]$ or be undefined ($\dagger$). The integers $lb(sv), ub(sv) \in \mathbb{Z}$ are called the lower and upper bounds of the state variable. The set of numeric state variables is denoted as $\mathcal{SV}_{\mathcal{N}}$.

Remark. While our formalism uses integers for numeric state variables, this is rarely limiting as values can be scaled to appropriate precision as common in CSP solvers.

Definition 2 (State)**.** A *state* is the representation of the environment at a given time, defined as a total function mapping every ground (variable-free) state variables to an appropriately typed value.

The state variables can be used to build more complex expressions that will be used in subsequent definitions.

Definition 3 (Symbolic Expression)**.** A *symbolic expression* is a symbol that can be evaluated to an object in $\mathcal{O}$ or be undefined. It can be either:

- A constant symbol from the set of objects: $o \in \mathcal{O}$.
- A symbolic parameter: p with a domain in $\mathcal{O}' \subseteq \mathcal{O}$.
- A symbolic state variable: $sv(x) \in \mathcal{SV}_{\mathcal{S}}$.

Definition 4 (Numeric Expression)**.** A *numeric expression* is a linear sum that can be evaluated to an integer value in $\mathbb{Z}$ or be undefined. It is of the form $\omega_0 + \sum_i \omega_i \cdot sv_i(x_i)$ where $\omega_0, \omega_i \in \mathbb{Z}$ are constant integer coefficients and $sv_i(x_i) \in \mathcal{SV}_{\mathcal{N}}$ are numeric state variables.

Definition 5 (Boolean Expression)**.** A *boolean expression* is a proposition that can be evaluated to true (it is called *satisfied*), false (*unsatisfied*), or undefined. It can be either:

- A boolean atom: true ($\top$) or false ($\bot$).
- A boolean state variable: $sv(x) \in \mathcal{SV}_{\mathcal{B}}$.
- The comparison of a numeric expression N with zero: $N \bowtie 0$, where $\bowtie \in \{ <, \leq, >, \geq, =, \neq \}$.
- The comparison of two symbolic expressions S_1 and S_2: $S_1 \mathrel{\mathring{=}} S_2$, with $\mathring{=} \in \{ =, \neq \}$.
- A propositional formula of other boolean expressions.

For the three types of expressions, the evaluation of the expression E at a given time-point $t \in \mathcal{T}$ is done by substituting the state variables by their value at t and is denoted $[t]E$. It is said *undefined* if at least one used state variable is undefined at this time. Finally, the expression is said to be *parametrized* if it contains at least one variable parameter.

Definition 6 (Effect)**.** *Effects* are divided into two categories:

- An *assign effect* $[t]sv(x) := v$ on a state variable $sv(x)$ is the assignment of a new value $v \neq \dagger$ with a compatible type to this state variable.
- An *increase effect* $[t]sv(x) \mathrel{+}= N$ on a numeric state variable $sv(x) \in \mathcal{SV}_{\mathcal{N}}$ is the increase of the current value of $sv(x)$ by the value of a numeric expression N. If $sv(x)$ is undefined, the increase effect cannot be applied.

Both effect are said to be executed at a time $t \in \mathcal{T}$. As in PDDL2.1, the effect will only establish the value at the next time-point $t + \varepsilon$.

Definition 7 (Action template)**.** An *action template* describes an operation that can be performed in the current environment, defined by the tuple $a = (\, name_a, P_a, s_a, e_a, C_a, X_a, E_a \,)$:

- $name_a$ is the unique name of the action.
- $P_a = [\, p_a^1, \ldots, p_a^n \,]$ is its list of variable parameters.
- $s_a, e_a \in \mathcal{T}$ are the start and end variable time-points.
- X_a is its set of constraints, each one being a time-independent boolean expression parametrized by $P' \subseteq P_a$. It is common practice to have a constraint on the duration, for instance: $e_a - s_a = k \cdot \varepsilon$, for some $k \in \mathbb{N}$.
- C_a is its set of conditions, each one being a time-dependent boolean expression parametrized by $P' \subseteq P_a$ which must be satisfied at a time-point $t \in [\![s_a, e_a]\!]$.
- E_a is its set of effects, each one being parametrized by $P' \subseteq P_a$ and executed at a time-point $t \in [\![s_a, e_a]\!]$.

An action template is said to be *ground* when its variable parameters have been replaced by symbols in $\mathcal{O}$, and its start and end time-points by constants in $\mathcal{T}$.

Definition 8 (Planning Problem)**.** A *planning problem* is represented by the tuple $\mathcal{P} = (\, \mathcal{O}, \mathcal{SV}, A, s_0, G \,)$ where:

- $\mathcal{O}$ is the set of symbols representing the objects.
- $\mathcal{SV} = \mathcal{SV}_{\mathcal{B}} \cup \mathcal{SV}_{\mathcal{N}} \cup \mathcal{SV}_{\mathcal{S}}$ is the set of state variables.
- A is the set of action templates.
- s_0 is the initial state, a partial assignment to state variables $\mathcal{SV}$ at the temporal origin 0.
- G is the set of goals, a set of boolean expressions which must be satisfied at the horizon H.

From a planning problem, the objective for an automated planner is to derive a plan Π, *i.e.*, a set of ground actions whose sequenced effects will update the state from s_0 to a final state where all goals hold.

The set of effects of a plan Π is denoted as E_Π and is composed of the effects of all actions in the plan. The subset of effects that apply at time t on the state variable $sv(x)$ is denoted as $E_\Pi^{[t]sv(x)}$. Because a state variable may only assume a single value at a given time-point, two assign effects cannot simultaneously change the value of the same state variable.

A plan Π is said to be *coherent* if all its effects are coherent, *i.e.*, for any $t \in \mathcal{T}$ and $sv(x) \in \mathcal{SV}$:

$$E_\Pi^{[t]sv(x)} = \begin{cases} \emptyset & \text{is empty} \\ \{ := v \} & \text{has only one assign} \\ \{ \mathrel{+}= N_i \;\mid\; 1 \leq i \leq k \} & \text{has } k \in \mathbb{N} \text{ increases} \end{cases}$$

Definition 9 (Trace)**.** A trace is a time-indexed sequence of states that represents the evolution of the environment at each time-point. Given a coherent plan Π, the evolution over time of a state variable $sv(x)$ is recursively computed from its initial value $[0]sv(x)$, defined in the initial state s_0, as follows:

$$[t+\varepsilon]sv(x) = \begin{cases} [t]sv(x) & \text{if } E_{\Pi}^{[t]sv(x)} = \emptyset \\ v & \text{if } E_{\Pi}^{[t]sv(x)} = \{ := v \} \\ [t]sv(x) + \sum_i N_i & \text{if } E_{\Pi}^{[t]sv(x)} = \{ += N_i \} \end{cases}$$

Remark. Increase effects are cumulative when applied simultaneously, matching scheduling solvers [28] but differing from PDDL2.1.

Definition 10 (Solution Plan)**.** A plan Π is a *solution* of $\mathcal{P}$ iff the following conditions are met: (i) Π is coherent. (ii) Numeric state variables are always within their domain or can be undefined. (iii) All actions in Π are applicable, *i.e.*, all their conditions are satisfied by the trace. (iv) All goals are satisfied at the horizon H. (v) All effects in the plan are executed before H.

Definition 11 (Quality)**.** A problem can have multiple solutions that can be compared with a *quality metric*, that is either:

- The number of actions in the plan: $|\Pi|$.
- The duration of the plan (*makespan*): $\max \{e_a | a \in \Pi\}$.
- The value of a numeric expression N at the horizon.

Without loss of generality, we will focus only on minimization.

III. Planning as Constraint Solving

To solve such a planning problem, we adapt the pure-temporal encoding of LCP [23] — which compiles the problem into a series of Constraint Satisfaction Problems (CSPs) solved by a dedicated solver — to our action-based formalism with support for numeric features. The CSP encoding contains both boolean and integer variables, with logical and arithmetic constraints handled through branch-and-bound search and clause learning in the underlying Aries CP solver [29].

CP solvers require a fix-sized encoding in which all variables are known a priori. Just like SAT-based planners, LCP relies on a variable $k \in \mathbb{N}$ that bounds the size of the solution plans currently considered.

Algorithm 1 presents the main loop of the solver, adapted from LCP to support optimization. For a given bound $k \in \mathbb{N}$, the problem is encoded into a CSP $(\mathcal{V}_k, \mathcal{X}_k)$, where $\mathcal{V}_k$ is the set of variables and $\mathcal{X}_k$ is the set of constraints (l. 4). If a solution is already known, an additional constraint imposes an improvement to the cost (l. 6). A CSP solver is invoked to find a satisfying assignment (l. 7). If the solver returns UNSAT, more actions are allowed for the next iteration (l. 9). If a satisfying assignment is found, the corresponding plan is shared with the caller (l. 12), and the search continues with the same bound k with the aim of improving the quality.

This algorithm is able to prove the optimality of the solution only within a given maximum bound $k_{\max}$. Completeness can be maintained by setting $k_{\max}$ to ∞ (the default).

Algorithm 1 Main algorithm for a planning problem $\mathcal{P}$, producing a sequence of plans of increasing quality within a given maximum bound $k_{\max}$.

```
1:  bestPlan ← ∅
2:  k ← 0
3:  while k ≤ k_max do
4:      (V_k, X_k) ← encode(P, k)
5:      if bestPlan ≠ ∅ then ▷ Require quality improvement
6:          X_k ← X_k ∪ {cost(V_k) < cost(bestPlan)}
7:      assignment ← solveCsp(V_k, X_k)
8:      if assignment = UNSAT then
9:          k ← k + 1                          ▷ Allow more actions
10:     else
11:         bestPlan ← extractPlan(assignment)
12:         yield bestPlan                     ▷ Notify of new plan
```

a) Bounded Problem: The bound k determines the size of the allowed solution. As in LCP, for a given bound k, the solution plan may contain at most k instances of an action. For example, given a bound $k = 2$, a solution to a *Blocksworld* problem may contain at most 2 *pickup* actions, at most 2 *putdown* actions, and so on. For a given $k \in \mathbb{N}$, we build the set of action instances $\mathbb{A}_k$ by instantiating each action $a \in A$ into k action instances: $\mathbb{A}_k = \bigcup_{a_i \in A} \{ a_i^1, \dots, a_i^k \}$. Each action instance in $\mathbb{A}_k$, represents a potential action of the plan whose participation to the solution as well as its actual parameters and start/end times remain to be determined by the CSP solver.

b) Plan extraction: As we will detail in the following sections, the set of variables $\mathcal{V}_k$ notably contains for each action instance $a \in \mathbb{A}_k$: a boolean variable $pres(a)$ that is true iff a appears in the solution plan, one variable for each parameter of a, and two variables respectively representing the start and end times of a. From an assignment ψ to variables $\mathcal{V}_k$ produced by a CSP solver, one can thus trivially determine: which action instances participate in the plan as well as their parameters and execution times. More formally the corresponding plan is defined as $\Pi_\psi = \{ \psi(a) \mid a \in \mathbb{A}_k \text{ s.t. } \psi \models pres(a) \}$ where $\psi(a)$ denotes the ground action obtained through the substitution of the parameters and time-points of a by their value in ψ.

A. Tokens for Capturing State Evolution

Our problem definition contains several structures that may have similar interpretation despite their different origin. For instance, assignment to state variables may be found in the initial state, and in action instances.

This section introduces the core elements used to provide a unified view of such structures regardless of their origin. These are named *tokens* after the eponymous elements that serve a similar role in timeline-based planning [21]. We distinguish four kinds of tokens, respectively representing (i) the evaluation of a state variable at a given time, (ii) a unified time-independent view of constraints, conditions, and goals, (iii) a unified view of the application of an assignment,

and the initial state, and (iv) the application of an increase effect. They all are associated with the presence variable of the associated action introduced earlier, so the different structures are standalone.

1) Read Token (new contribution): A read token is used to reify the value of a state variable at a given time and is of the form: $p : [t]sv(x) = v$. The token states that, if its p is true, then the variable v must take the value of the state variable $sv(x)$ at time t. These are primarily used to simplify the problem definition by replacing (nested) expressions with fresh variables. For instance, a condition $[t]\mathit{free}(\mathit{loc}(r))$ appearing in an action instance a would be simplified by introducing two read tokens $\mathit{pres}(a) : [t]\ \mathit{loc}(r) = \ell$ and $\mathit{pres}(a) : [t]\ \mathit{free}(\ell) = f$, where ℓ and f are two new variables introduced to store the value of the corresponding state variables. Having these two tokens, the condition is now equivalent to stating that f should evaluate to true.

Each time a (potentially nested) state variable expression appears in the problem, a new read token is introduced, and the state variable is replaced by the new variable. The set of read tokens T_R is built from the state variables value used in the action instances, the goals, and the plan quality objective:

$$\begin{aligned} T_R = &\{\ \mathit{pres}(a) : [t]sv(x) = v \ \mid\ a \in \mathbb{A}, sv(x) \text{ used in } a\ \} \\ &\cup \{\ \top : [H]sv(x) = v \ \mid\ g \in G, sv(x) \text{ used in } g\ \} \\ &\cup \{\ \top : [H]sv(x) = v \ \mid\ sv(x) \text{ used in the quality } N\ \} \end{aligned}$$

Note that the scope p limits the reach of tokens. In particular, it specifies that a token originating from an action instance a only needs to be considered when $\mathit{pres}(a)$ holds, while a read token originating from a goal should always be considered.

2) Condition Token (adapted from LCP*):* Each boolean expression B obtained by replacing the state variables by their "read" variables in constraints and conditions is associated with a condition token: $p : B$. The token states that, if it is present ($p = \top$), then B must be satisfied. The set of condition tokens T_C is built from the boolean expressions used in the conditions and constraints of the action instances, and the goals:

$$\begin{aligned} T_C = &\{\ \mathit{pres}(a) : c \ \mid\ a \in \mathbb{A}, c \in C_a\ \} \\ &\cup \{\ \mathit{pres}(a) : c \ \mid\ a \in \mathbb{A}, c \in X_a\ \} \\ &\cup \{\ \top : g \ \mid\ g \in G\ \} \end{aligned}$$

3) Assign Effect Token (adapted from LCP*):* Each assign effect $[t]sv(x) := v$ of an action instance is associated with an assign effect token: $p : [t, m]sv(x) := v$. The token states that, if it is present, then $sv(x)$ must be assigned the value v immediately after t. Moreover, the time-point variable m (with domain $\mathcal{T}$) is introduced for each assign effect to enforce a mutual exclusion (*mutex*) period $[t, m]$ where $sv(x)$ cannot be assigned a new value by another assign effect. This will be useful to ensure the coherence of the plan. The set of assign effect tokens T_E^A is built from the initial state, and the assign effects used in the action instances:

$$\begin{aligned} T_E^A = &\left\{ \begin{array}{l} \top : [0, m]sv(x) := v \\ \quad \mid v \text{ is the initial value of } sv(x) \end{array} \right\} \\ &\cup \left\{ \begin{array}{l} \mathit{pres}(a) : [t, m]sv(x) := v \\ \quad \mid a \in \mathbb{A}, [t]sv(x) := v \in E_a \end{array} \right\} \end{aligned}$$

4) Increase Effect Token (new contribution): Each increase effect $[t]sv(x) \mathrel{+}= N$ of an action instance is associated with an increase effect token: $p : [t]sv(x) \mathrel{+}= N$. The token states that, if it is present, then $sv(x)$ must be increased by the value of N immediately after time t. The set of increase effect tokens T_E^I is built from the increase effects used in the action instances:

$$T_E^I = \left\{ \begin{array}{l} \mathit{pres}(a) : [t]sv(x) \mathrel{+}= N \\ \quad \mid a \in \mathbb{A}, [t]sv(x) \mathrel{+}= N \in E_a \end{array} \right\}$$

B. Variables used over State Progression

The set of variables $\mathcal{V}_k$ used to build the constraints of the CSP are the following:

$$\begin{aligned} \mathcal{V}_k = &\{\ \mathit{pres}(a)\ \mid\ a \in \mathbb{A}_k\ \} && \text{presence of each action} \\ &\cup \{\ p_a^i\ \mid\ a \in \mathbb{A}_k, p_a^i \in P_a\ \} && \text{parameters of each action} \\ &\cup \{\ s_a\ \mid\ a \in \mathbb{A}_k\ \} && \text{start of each action} \\ &\cup \{\ e_a\ \mid\ a \in \mathbb{A}_k\ \} && \text{end of each action} \\ &\cup \{\ v_\kappa\ \mid\ \kappa \in T_R\ \} && \text{value of each read token } \kappa \\ &\cup \{\ m_\alpha\ \mid\ \alpha \in T_E^A\ \} && \text{mutex of each assign token } \alpha \end{aligned}$$

C. Constraints over State Progression

The constraints are used to ensure the consistency of the solution plan, *i.e.*, the constraints and conditions are satisfied, and the effects are coherent. Some constraints are adapted from LCP [23] to support our action-based formalism: (i) the support constraint, (ii) the coherence constraint, and (iii) a part of the structure constraints. They are given for completeness of the paper.

1) Support Constraint (adapted from LCP*):* An assign effect token $\alpha \in T_E^A$ is said to *support* a non-numeric read token $\kappa \in T_R$ with $sv(x) \in \mathcal{SV} \setminus \mathcal{SV}_{\mathcal{N}}$ if it assigns a value to $sv(x)$ and maintains this value during the evaluation.

$$\text{Given } \left\{ \begin{array}{lll} \kappa & = \langle\ p_\kappa : [t_\kappa]sv(x_\kappa) = v_\kappa\ \rangle & \in T_R \\ \alpha & = \langle\ p_\alpha : [t_\alpha, m_\alpha]sv(x_\alpha) := v_\alpha\ \rangle & \in T_E^A \end{array} \right.$$

The constraint *supported-by* (κ, α) is defined as:

$$p_\alpha \wedge t_\alpha < t_\kappa \wedge t_\kappa \leq m_\alpha \wedge x_\kappa = x_\alpha \wedge v_\kappa = v_\alpha$$

The read token κ is said to be *supported* when it is present and there exists an assign effect token α that supports it. Formally, the constraint *supported* (κ) is defined as:

$$p_\kappa \implies \bigvee_{\alpha \in T_E^A} \textit{supported-by}\,(\kappa, \alpha)$$

Remark. $x = \left[\, x^1, \ldots, x^n \,\right]$ is a list of variables, therefore, $x_1 = x_2$ is a shorthand for $\bigwedge_{i=1}^{n} x_1^i = x_2^i$.

2) Numeric Support Constraint (new contribution): An assign effect token $\alpha \in T_E^A$ and a set of increase effect tokens $\Phi \subseteq T_E^I$ are said to *support* a numeric read token $\kappa \in T_R$ with $sv(x) \in \mathcal{SV}_\mathcal{N}$ if:

- α assigns a value to $sv(x)$ and maintains its mutex period until the evaluation time.
- Each $\varphi_i \in \Phi$ increases the value of $sv(x)$ after α execution and before the evaluation time.

$$\text{Given} \begin{cases} \kappa = \langle\, p_\kappa : [t_\kappa] sv(x_\kappa) = v_\kappa \,\rangle & \in T_R \\ \alpha = \langle\, p_\alpha : [t_\alpha, m_\alpha] sv(x_\alpha) := v_\alpha \,\rangle & \in T_E^A \\ \Phi = \{\, \varphi_i = \langle\, p_\varphi^i : [t_\varphi^i] sv(x_\varphi^i) \mathrel{+}= N_\varphi^i \,\rangle \,\} & \subseteq T_E^I \end{cases}$$

Two intermediates variables are introduced to define the numeric support constraint. Firstly, the variable $\chi_\alpha(\kappa, \alpha)$ is constrained to be true iff α assigns a value to $sv(x_\kappa)$ and maintains its mutex period until the evaluation time:

$$\chi_\alpha(\kappa, \alpha) \iff (p_\alpha \wedge t_\alpha < t_\kappa \wedge t_\kappa \leq m_\alpha \wedge x_\kappa = x_\alpha)$$

Secondly, the variable $\chi_\varphi(\kappa, \alpha, \varphi_i)$ is constrained to be true iff φ_i increases the value of $sv(x_\kappa)$ after the execution of α and before the evaluation time:

$$\chi_\varphi(\kappa, \alpha, \varphi_i) \iff \left(p_\varphi^i \wedge t_\alpha < t_\varphi^i \wedge t_\varphi^i < t_\kappa \wedge x_\kappa = x_\varphi^i\right)$$

Lastly, the constraint *num-supported-by* (κ, α, Φ) is:

$$\begin{aligned} & \chi_\alpha(\kappa, \alpha) \wedge lb(sv) \leq v_\kappa \wedge v_\kappa \leq ub(sv) \\ \wedge \quad & v_\kappa = v_\alpha + \textstyle\sum_{i=1}^{n} \left[\chi_\varphi(\kappa, \alpha, \varphi_i)\right] \cdot v_\varphi^i \end{aligned}$$

Where $[P]$ is 1 if P is true, 0 otherwise. This effectively cancels the contribution of any irrelevant increase token φ_i.

Finally, κ is said to be *numerically supported* when it is present and if there exists an assign effect token and a set of increase effect tokens that numerically support it. Formally, the constraint *num-supported* (κ) is defined as:

$$p_\kappa \implies \bigvee_{\alpha \in T_E^A} \textit{num-supported-by}\left(\kappa, \alpha, T_E^I\right)$$

Figure 1 illustrates the numeric support of a state variable *energy*. It shows in green that the read tokens ($\geq$ 20?) are supported by the latest *refuel* action and the consumptions of the *travel* actions that occur since the *refuel* (and prior to the read condition).

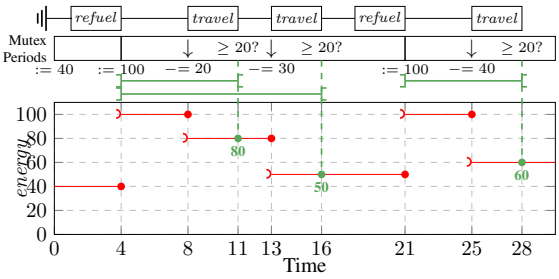

Fig. 1: Evolution of an *energy* state variable over time, initially set to 40. The *refuel* action sets the value to 100 at the end. The *travel* action consumes some *energy* at the start and checks the value at the end.

3) Coherence Constraint (adapted from LCP*):* Two distinct assign effect tokens are said to be *coherent* if they do not change the value of the same state variable at the same time.

$$\text{Given} \begin{cases} \alpha_1 = \langle\, p_1 : [t_1, m_1] sv(x_1) := v_1 \,\rangle & \in T_E^A \\ \alpha_2 = \langle\, p_2 : [t_2, m_2] sv(x_2) := v_2 \,\rangle & \in T_E^A \end{cases}$$

The constraint *coherent* (α_1, α_2) is defined as:

$$p_1 \wedge p_2 \implies (m_1 \leq t_2 \vee m_2 \leq t_1 \vee x_1 \neq x_2)$$

Figure 1 also illustrates the mutex periods of the assign effect tokens, represented by the "Mutex Periods" rectangles, and their coherence, represented by the absence of overlap.

Similarly, one assign effect token and one increase effect token are said to be *coherent* if they do not update the same state variable at the same time.

$$\text{Given} \begin{cases} \alpha = \langle\, p_\alpha : [t_\alpha, m_\alpha] sv(x_\alpha) := v_\alpha \,\rangle & \in T_E^A \\ \varphi = \langle\, p_\varphi : [t_\varphi] sv(x_\varphi) \mathrel{+}= N_\varphi \,\rangle & \in T_E^I \end{cases}$$

The constraint *coherent* (α, φ) is defined as:

$$p_\alpha \wedge p_\varphi \implies (t_\varphi \neq t_\alpha \vee x_\alpha \neq x_\varphi)$$

4) Consistency Constraint (from LCP*):* A condition token $\beta \in T_C$ is said to be *consistent* if the boolean expression B is satisfied when the token is present. Given $\beta = \langle\, p : B \,\rangle \in T_C$, the constraint *consistent* (β) is defined as: $p \implies B$

5) Structure Constraint (partially adapted from LCP*):* Additional constraints are needed to ensure the structure of the actions and effects, *i.e.*, the relation between the start and end time-points of an action, the execution and the mutex (for assignments) time-points of an effect, and the horizon.

$$\text{Given} \begin{cases} a = (\, name_a, P_a, s_a, e_a, C_a, X_a, E_a \,) & \in \mathbb{A} \\ \alpha = \langle\, p_\alpha : [t_\alpha, m_\alpha] sv(x_\alpha) := v_\alpha \,\rangle & \in T_E^A \\ \varphi = \langle\, p_\varphi : [t_\varphi] sv(x_\varphi) \mathrel{+}= N_\varphi \,\rangle & \in T_E^I \end{cases}$$

The structure constraints are defined as:

$$\begin{aligned} \textit{structure-action}\,(a) : \textit{pres}(a) &\implies 0 \leq s_a \leq e_a \wedge e_a \leq H \\ \textit{structure-assign}\,(\alpha) : \quad p_\alpha &\implies t_\alpha < H \wedge t_\alpha < m_\alpha \\ \textit{structure-increase}\,(\varphi) : \quad p_\varphi &\implies t_\varphi < H \end{aligned}$$

D. Complete Encoding & Properties

The CSP can finally be built from the set of variables $\mathcal{V}_k$ and the set of constraints $\mathcal{X}_k$ introduced previously:

$$\begin{aligned} \mathcal{V}_k = & \{\, \textit{pres}(a) \mid a \in \mathbb{A}_k \,\} \cup \{\, p_a^i \mid a \in \mathbb{A}_k, p_a^i \in P_a \,\} \\ & \cup \{\, s_a \mid a \in \mathbb{A}_k \,\} \cup \{\, e_a \mid a \in \mathbb{A}_k \,\} \\ & \cup \{\, v_\kappa \mid \kappa \in T_R \,\} \cup \{\, m_\alpha \mid \alpha \in T_E^A \,\} \\ \mathcal{X}_k = & \{\, \textit{supported}\,(\kappa) \mid \kappa \in T_R, sv(x_\kappa) \in \mathcal{SV} \setminus \mathcal{SV}_\mathcal{N} \,\} \\ & \cup \{\, \textit{num-supported}\,(\kappa) \mid \kappa \in T_R, sv(x_\kappa) \in \mathcal{SV}_\mathcal{N} \,\} \\ & \cup \{\, \textit{coherent}\,(\alpha_1, \alpha_2) \mid \alpha_1, \alpha_2 \in T_E^A, \alpha_1 \neq \alpha_2 \,\} \\ & \cup \{\, \textit{coherent}\,(\alpha, \varphi) \mid \alpha \in T_E^A, \varphi \in T_E^I \,\} \\ & \cup \{\, \textit{consistent}\,(\beta) \mid \beta \in T_C \,\} \\ & \cup \{\, \textit{structure-action}\,(a) \mid a \in \mathbb{A}_k \,\} \\ & \cup \{\, \textit{structure-assign}\,(\alpha) \mid \alpha \in T_E^A \,\} \\ & \cup \{\, \textit{structure-increase}\,(\varphi) \mid \varphi \in T_E^I \,\} \end{aligned}$$

These definitions complete the encoding step of Algorithm 1. The symmetry breaking constraints of LCP [23] may

be added, but are omitted for the sake of space. Also omitted are the straightforward constraints for binding the cost of a solution to a numeric variable (Def. 11).

Theorem 1 (Completeness)**.** *If the planning problem has a solution, then Algorithm 1 will return a solution plan.*

Proof (Sketch). For a given solution plan Π, we show that there exists a procedure to determine a bound k and an assignment to $\mathcal{V}_k$ such that all constraints $\mathcal{X}_k$ are satisfied, and the assignment represents the plan Π.[2] □

Theorem 2 (Soundness)**.** *All plans produced by Algorithm 1 are valid with respect to Def. 10.*

Proof (Sketch). For each of the requirements of Def. 10, one can show that there is a corresponding set of constraints in the CSP whose satisfaction implies that the extracted plan will uphold the requirement. □

It follows from Theorem 1 that Algorithm 1 will eventually find the optimal plan if the instance is satisfiable. However, Algorithm 1 only proves optimality with respect to a bound k on the number of actions. Automatically computing such upper bounds remains a difficult task [30] and to the best of our knowledge not studied for the numeric and temporal classes of automated planning we consider. In our experience, it is generally easy to derive tight domain-specific bounds when facing a particular problem.

IV. Experiments

A. ARIES: Planner Implementation

For a given planning problem, defined in the PDDL format, we use the *unified-planning* library [31] for parsing the problem into a format compatible to our formalism. The planner works by encoding the planning problem into a CSP with the corresponding difference, equality, conjunctive and disjunctive constraints within a Lazy Clause Generation CP solver, specialized for scheduling [29]. The high-level loop proceeds as in Algorithm 1, by increasing the bound k on the number of replicated actions, starting at 0. Once the encoded problem is shown unsatisfiable or the current solution proven optimal, the bound is incremented by 1 and process continues. In practice, $k_{\max}$ is set to infinity, allowing the solver to explore unbounded solution spaces.

For each bound, the solver is configured to run an activity-based search (VSIDS), initially restricted to boolean variables. When optimizing, value selection leverages solution-guidance [29], where the CP solver will favor the value each variable has in the best known solution.

B. Planners

We compare the performance of ARIES with two state-of-the-art automated planners that support PDDL2.2.

OPTIC [12] is the most established planner in the space of temporal and numeric planning. Based on forward state-space search, partially-ordered plans are built incrementally from the initial state, using a best-first search strategy guided by a delete-relaxation heuristic. Consistency of numeric and temporal variables are handled in a MILP program incrementally updated at each node of the search graph.

LPG [32] is a hybrid planner combining a state-space search for totally ordered plans and a local-search over partially-ordered plans, both guided by delete-relaxation heuristics. While LPG in general shows outstanding performance, one should keep in mind that it is incomplete with respect to the semantics of PDDL2.2 as its internal representation does not support *required-concurrency* where two actions necessarily overlap (as in the *Match-Cellar* domain of our benchmarks) [33].

C. Domains

For evaluation, we consider six temporal-numeric domains from past International Planning Competitions (IPC) and two adaptations of established scheduling problems.

1) IPC: Depots, *Rovers*, and *Satellite* are IPC 2002 benchmarks with real numerics rounded to integers and pre-computations for division. *Openstacks* and *UMTS* are from IPC 2008 and 2004 respectively. *Match-Cellar* is a numeric IPC 2011 variant using numeric state variables for matches and fuses.

2) Scheduling: We provide scheduling instances as PDDL2.1. *JobShop with Operators* and *RCPSP* optimize makespan with precedence and resource constraints. Action-selection is trivial (all operations are required) but tests temporal/resource optimization. Each activity appears exactly once, so $k = 0$ has no solutions and the solver starts with $k = 1$.

JobShop uses Lawrence [34] instances (3-5 operators). *RCPSP* uses PSPLib [35] `j30`/`j60`/`j120` instances with increase effects for resources. Both use PDDL2.1 with predicates forcing activity presence and precedence, with resource usage as increase effects on numeric state variables.

D. Metrics

The planners are run in an anytime configuration, *i.e.*, returning plans of increasing quality, over a 600-second period. They are evaluated based on the following metrics.

The **Coverage** (`Cov`) is the percentage of instances for which at least one solution was returned.

The **IPC** score is the one of the satisficing track of the International Planning Competition (IPC) and favors high-quality plans. The score for a solved instance is the ratio between the quality of the returned plan and the best quality of any returned plan by any planner for this instance. Quality computation depends on the problem's metric: for makespan optimization, it is the plan's total duration; for numeric objectives, it is the value of the specified numeric expression. For unsolved instances, it is 0. The final score is the mean of all the previous scores, multiplied by 100.

E. Results and Discussion

The planners were run on a cluster of machines with Intel E5-2695 v3 2.3G CPUs, with one CPU per planner,

[2] A detailed proof is available at https://github.com/plaans/resources-ictai-25

	OPTIC		LPG		ARIES		Δ Portfolio ↗	
	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC
Depots (22)	54.55	28.05	**95.45**	**92.90**	54.55	52.65	0.00	2.55
Jobshop (40)	**100**	45.89	**100**	73.21	**100**	**93.38**	0.00	23.81
Match Cellar (20)	**100**	**83.93**	0.00	0.00	80.00	80.00	0.00	13.74
Openstacks (30)	**100**	**98.08**	**100**	39.62	83.33	63.43	0.00	1.85
Rcpsp (30)	**100**	59.90	**100**	86.93	**100**	**100.00**	0.00	13.07
Rovers (20)	55.00	37.24	**100**	**97.20**	55.00	47.88	0.00	1.75
Satellite (20)	75.00	66.26	**100**	**83.40**	95.00	62.62	0.00	1.97
Umts (50)	**100**	97.50	**100**	**100.00**	**100**	74.24	0.00	0.00
Average	85.57	64.61	**86.93**	71.66	83.48	**71.78**	0.00	7.34

TABLE I: Performances comparison of the different planners on the different domains. Δ Portfolio indicates the performance increase of the portfolio when ARIES is added.

a limitation of 16GB of memory, a 600 seconds wall-time timeout, and a temporal precision of 0.01 (*i.e.*, $\mathcal{T} = \{ 0, 0.01, 0.02, \ldots, H \}$). Problem instances and detailed results are available in an online appendix.[3]

The Table I depicts the performance of the individual planners on the different domains. A first challenge for the solvers is to find a valid plan regardless of its quality. LPG is the most performant at this task, finding at least one valid plan for all problems but one instance of *Depots* and all instances of *Match-Cellar* (because of its inability to handle required-concurrency [33]). While this translates to a very high coverage, LPG also produces plans of good quality, comparable to ARIES. On the other hand, OPTIC shows a good coverage but produces plans of lower quality, on average about 7.17% worse than the best alternatives.

ARIES achieves the best plan quality on average, with LPG performing very similarly. A deeper look at the results show different strengths however. On most problems, OPTIC finds quickly a first solution of reasonable quality, which is then only slowly improved as time passes. On the other hand, ARIES in general requires more time to provide a first solution, often of worse quality. From this point on however, the solution quality tends to improve steadily and rapidly. This is somewhat expected from the absence of planning-specific heuristics in the search, until the discovery of the first plan where the solution-guided systematic exploration of the search space works well in practice [29].

A detailed analysis shows that ARIES spends most of its CPU time in proving unsatisfiability or optimality for a given bound k, before proceeding to the $k + 1$ bound. This is particularly true for the bound immediately preceding the one where a first solution lies. This is unsurprising as, at this bound, the absence of a solution is not obvious and proving it may be challenging. A similar behavior has been noticed in SAT-based planners for classical planning [14].

To better depict the contribution of ARIES to the planning ecosystem, the right part of the table shows the increase in coverage and plan quality that would result in having ARIES in a portfolio as compared to just LPG and OPTIC. This portfolio operates as a Virtual Best Solver (VBS), running the three planners (ARIES, OPTIC, LPG) in parallel and returning the best plan found among them. The coverage does not increase, meaning that ARIES does not bring new solutions to instances where none existed. However, the average plan quality increases by 7.34%. This is most notable for the scheduling problems but the contribution is significant in most domains.

[3]https://github.com/plaans/resources-ictai-25

V. WARM STARTING

As seen in the previous experimental results, finding an initial solution for ARIES is in general hard and time consuming, but once a first solution is found it can be rapidly improved. This is a common characteristic of combinatorial optimization solvers based on constraint programming and many of them provide the ability to *warm-start* the solver with an initial solution. The initial solution provides upper bounds on the objective and an assignment to all decision variables that can be used by branching heuristics. While warm-starting is often done with a domain-specific greedy algorithm, in this section we propose to explore the benefits of warm-starting ARIES with the solution provided by other automated planners which allows to maintain domain-independence while benefiting from their velocity at finding initial solutions.

To run a warm-started ARIES_X solver with an initial providing solver X, we start by running the solver X until it finds its first solution. It is then immediately stopped and the remaining time is allocated to ARIES, with the plan as input.

Exploiting an initial *warm-start* plan in ARIES is a two steps process. First, we force ARIES to exactly replicate the initial plan. This is done by (i) creating a bounded problem with exactly one action instance per action in the initial plan, and (ii) adding constraints that force each action of the initial plan to match an action instance in the CSP solution (*i.e.*, same parameter and timepoints values). The purpose of this step is to obtain a complete assignment over decision variables in the bounded CSP. Once this first step is completed, the constraints forcing the initial solution are retracted and the solver proceeds normally, *i.e.*, using the previous solution for guidance and increasing the number of actions in the bounded problems once optimality is proven for the current bound.

A. Experiments

Table II shows the increase in coverage and plan quality of OPTIC and LPG when their first plan is optimized by ARIES.

Coverage increases only on *Match-Cellar* for $\text{ARIES}_{\text{LPG}}$ (due to required-concurrency handling), confirming that ARIES does not bring new resolutions compared to the original planner. However, the average plan quality increases by 10.89% for OPTIC, and 5.81% for LPG. This shows that, as expected, ARIES is able to greatly optimize plans when given a starting point. This result confirms and generalizes previous findings on a domain-specific problem [36].

VI. CONCLUSION

We presented a new lifted formalism for numeric and temporal planning with a planner that adapts an existing

	OPTIC		ARIES$_{\text{OPTIC}}$		LPG		ARIES$_{\text{LPG}}$	
	Cov	IPC	Cov	IPC	Cov	IPC	Cov	IPC
Depots (22)	54.55	28.05	54.55	38.34	**95.45**	**92.90**	**95.45**	81.73
Jobshop (40)	**100**	45.89	**100**	**91.38**	**100**	73.21	**100**	89.02
Match Cellar (20)	**100**	83.93	**100**	**93.06**	0.00	0.00	80.00	80.00
Openstacks (30)	**100**	**98.08**	**100**	96.99	**100**	39.62	**100**	59.40
Rcpsp (30)	**100**	59.90	**100**	81.57	**100**	**86.93**	**100**	84.35
Rovers (20)	55.00	37.24	55.00	35.55	**100**	**97.20**	**100**	60.17
Satellite (20)	75.00	66.26	75.00	69.34	**100**	**83.40**	**100**	66.67
Umts (50)	**100**	97.50	**100**	97.77	**100**	**100.00**	**100**	98.44
Average	85.57	64.61	85.57	75.50	86.93	71.66	**96.93**	**77.47**

TABLE II: Performances comparison of the different warm-started planners with their standalone counterparts.

CSP encoding for chronicles to our action-based formalism, extended with numeric planning and optimization support.

This results in an original planner in this area dominated by ground state-space search solvers. ARIES exploits a fully lifted representation and delegates *plan-space* exploration to a clause learning CP solver. Despite its originality, the solver is competitive on PDDL and scheduling benchmarks and excels at finding optimized solutions. Our results show ARIES's bottleneck is finding initial solutions, while it excels at optimization. A warm start approach with plans from other state-of-the-art planners mitigates this limitation, with resulting hybrid planners out-performing originals and competitive with standalone portfolios.

REFERENCES

[1] M. Ghallab and H. Laruelle, "Representation and Control in IxTeT, a Temporal Planner," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 1994.

[2] D. E. Smith, J. D. Frank, and A. K. Jónsson, "Bridging the gap between planning and scheduling," *The Knowledge Engineering Review*, 2000.

[3] J. S. Penberthy and D. S. Weld, "Temporal planning with continuous change," in *AAAI Conference on Artificial Intelligence*, 1994.

[4] V. Vidal and H. Geffner, "Branching and pruning: An optimal temporal pocl planner based on constraint programming," in *Artificial Intelligence*, 2004.

[5] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research (JAIR)*, 2001.

[6] M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research (JAIR)*, 2006.

[7] D. McDermott, M. Ghallab, A. Howe, C. A. Knoblock, A. Ram, M. Veloso, D. S. Weld, and D. Wilkins, "PDDL-the planning domain definition language," Tech. Rep., 1998.

[8] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *Journal of Artificial Intelligence Research (JAIR)*, 2003.

[9] M. B. Do and S. Kambhampati, "Sapa: a multi-objective metric temporal planner," *Journal of Artificial Intelligence Research (JAIR)*, 2003.

[10] A. Coles, A. Coles, M. Fox, and D. Long, "Colin: planning with continuous linear numeric change," *Journal of Artificial Intelligence Research (JAIR)*, 2012.

[11] K. Halsey, D. Long, and M. Fox, "CRIKEY - a temporal planner looking at the integration of scheduling and planning," in *ICAPS Workshop on Integrating Planning into Scheduling*, 2004.

[12] J. Benton, A. Coles, and A. Coles, "Temporal Planning with Preferences and Time-Dependent Continuous Costs," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[13] A. Valentini, A. Micheli, and A. Cimatti, "Temporal Planning with Intermediate Conditions and Effects," in *AAAI Conference on Artificial Intelligence*, 2020.

[14] J. Rintanen, "Planning as satisfiability: Heuristics," *Artificial Intelligence*, 2012.

[15] D. Höller and G. Behnke, "Encoding Lifted Classical Planning in Propositional Logic," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2022.

[16] M. F. Rankooh and G. Ghassem-Sani, "ITSAT: an efficient sat-based temporal planner," *Journal of Artificial Intelligence Research (JAIR)*, 2015.

[17] S. Panjkovic and A. Micheli, "Abstract action scheduling for optimal temporal planning via OMT," in *AAAI Conference on Artificial Intelligence*, 2024.

[18] F. Leofante, "Omtplan: A tool for optimal planning modulo theories," *Journal on Satisfiability, Boolean Modeling and Computation*, 2023.

[19] C. Piacentini, M. Castro, A. Cire, and J. C. Beck, "Compiling Optimal Numeric Planning to Mixed Integer Linear Programming," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

[20] E. Scala, M. Ramírez, P. Haslum, and S. Thiébaux, "Numeric planning with disjunctive global constraints via smt," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.

[21] J. Barreiro, M. Boyce, M. B. Do, J. Frank, M. Iatauro, T. Kichkaylo, P. H. Morris, J. Ong, E. Remolina, T. Smith, and D. E. Smith, "EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization," in *International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)*, 2012.

[22] A. Bit-Monnot, M. Ghallab, F. Ingrand, and D. E. Smith, "FAPE: a Constraint-based Planner for Generative and Hierarchical Temporal Planning," Tech. Rep., 2020.

[23] A. Bit-Monnot, "A Constraint-Based Encoding for Domain-Independent Temporal Planning," in *Principles and Practice of Constraint Programming*, 2018.

[24] D. Gnad, L.-o. Alon, E. Weiss, and A. Shleyfman, "Pdbs go numeric: Pattern-database heuristics for simple numeric planning," in *AAAI Conference on Artificial Intelligence*, 2025.

[25] X. Lin, Q. Chen, L. Fang, Q. Guan, W. Luo, and K. Su, "Generalized linear integer numeric planning," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2022.

[26] A. Shleyfman, R. Kuroiwa, and J. C. Beck, "Symmetry detection and breaking in linear cost-optimal numeric planning," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2023.

[27] S. Edelkamp and J. Hoffmann, "PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition," Tech. Rep., 2004.

[28] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG," *Constraints*, 2018.

[29] A. Bit-Monnot, "Enhancing Hybrid CP-SAT Search for Disjunctive Scheduling," in *European Conference on Artificial Intelligence (ECAI)*, 2023.

[30] M. Abdulaziz, "Plan-length bounds: Beyond 1-way dependency," in *AAAI Conference on Artificial Intelligence*, 2019.

[31] A. Micheli, A. Bit-Monnot, G. Röger, E. Scala, A. Valentini, L. Framba, A. Rovetta, A. Trapasso, L. Bonassi, A. E. Gerevini, L. Iocchi, F. Ingrand, U. Köckemann, F. Patrizi, A. Saetti, I. Serina, and S. Stock, "Unified planning: Modeling, manipulating and solving ai planning problems in python," *SoftwareX*, 2025.

[32] A. Gerevini, A. Saetti, and I. Serina, "An approach to efficient planning with numerical fluents and multi-criteria plan quality," *Artificial Intelligence*, 2008.

[33] W. Cushing, S. Kambhampati, Mausam, and D. S. Weld, "When is Temporal Planning Really Temporal?" in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[34] S. Lawrence, "Resource constrained project scheduling : an experimental investigation of heuristic scheduling techniques (supplement)," *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

[35] R. Kolisch and A. Sprecher, "PSPLIB - A project scheduling problem library: OR software - ORSEP Operations Research Software Exchange Program," *European Journal of Operational Research (EJOR)*, 1997.

[36] K. Booth, M. Do, J. Beck, E. Rieffel, D. Venturelli, and J. Frank, "Comparing and Integrating Constraint Programming and Temporal Planning for Quantum Circuit Compilation," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.