An Automata-Based Constraint Programming Framework for Optimal Classical Planning

Damien Van Meerbeeck
Linköping University, Sweden

Arnaud Lequen
Linköping University, Sweden

Gilles Pesant
Polytechnique Montréal, Canada

Jendrik Seipp
Linköping University, Sweden

Abstract

Recent work has shown that classical planning tasks can be compactly factored into deterministic finite automata and solved optimally with constraint programming (CP). In this setting, finding a plan reduces to finding a word accepted by all automata through Regular constraints. So far, however, these automata have had to be carefully handcrafted from PDDL tasks. In this paper, we show that they can instead be generated automatically and used as the basis of CP models. We also show that the resulting framework is easily extensible with additional constraints from the planning literature that strengthen propagation. Our approach solves more tasks than the state of the art in end-to-end CP for classical planning in almost all domains.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling

Keywords and phrases Optimal Classical Planning, Landmark Constraints, Automata

Digital Object Identifier 10.4230/LIPIcs.CP.2026.55

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.20081550` [42]

Funding This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Computations were performed at NSC Tetralith provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) funded by the Swedish Research Council through grant agreement no. 2022-06725.

1 Introduction

Classical planning is the challenge of finding a sequence of actions (called a *plan*) that transforms a given initial situation into one that satisfies a given goal description in a fully observable, deterministic environment [13]. In this paper, we consider *optimal* classical planning, where the goal is to find a plan with the fewest actions. Most of today's strongest optimal classical planners are based on state-space search with lower-bounding distance estimators, called *admissible heuristics* [17, 20, 36]. These heuristics guide the search through state spaces that are too large to enumerate explicitly.

In this work, we look beyond the state-space search paradigm and instead solve planning tasks using constraint programming (CP). Constraint programming provides a flexible language for combining declarative information and global constraints [35], which makes it attractive for planning models that need to integrate multiple sources of structure. In particular, CP makes it comparatively easy to enrich a planning encoding with additional constraints that strengthen propagation without changing the set of valid plans.

© Damien Van Meerbeeck, Arnaud Lequen, Gilles Pesant, and Jendrik Seipp;
licensed under Creative Commons License CC-BY 4.0

Our approach builds on recent automata-based representations of planning tasks. Starting from an input planning task, we automatically derive a set of deterministic finite automata that represent the dynamics of the task at the level of individual state variables, and use them as the backbone of a CP model. Finding a plan is then reduced to finding a word accepted by all automata, which we encode through Regular [28] constraints. Unlike previous automata-based CP approaches, which rely on manually engineered automata, our framework derives the atomic automata automatically from standard planning inputs such as PDDL [18]. This turns automata-based CP planning into a general pipeline rather than a domain-specific modeling task, and makes it possible to study preprocessing choices, model enhancements and redundant constraints systematically across a large benchmark set.

The dominance of heuristic search in optimal classical planning is largely due to the wide array of admissible heuristics that can be extracted from the planning model. Since CP provides a rich modeling language, we show how to directly encode the information underlying such heuristics in our framework as redundant constraints that strengthen propagation. Concretely, we focus on two families of constraints from the operator-counting framework [30]: landmark constraints and net change constraints.

Our experimental evaluation on benchmarks from the Optimal Tracks of the International Planning Competition shows that our framework outperforms CP-based baselines in almost all domains. Our results also show that the additional constraints can substantially strengthen propagation and improve solver performance, demonstrating that our approach is a promising framework for optimal classical planning.

The remainder of the paper is organized as follows. Section 2 discusses related work on compilation-based and CP-based approaches to planning. Section 3 reviews the planning setting and the automata-based factored representations that we build upon. Section 4 presents our automata-based CP encoding and overall solution procedure. Section 5 shows how we strengthen our model with operator-counting constraints. Section 6 details our experimental setup and Section 7 reports results. Section 8 discusses limitations and extensions, and Section 9 concludes.

2 Related Work

Optimal classical planning has been studied through several paradigms besides heuristic state-space search. The work most closely related to ours falls into three groups: compilation-based methods, constraint-based methods and approaches based on factored transition systems.

2.1 Compilation-Based Approaches

One influential line of work reduces bounded planning problems to other reasoning formalisms. Starting from the seminal planning-as-satisfiability formulation of Kautz and Selman [24], later SAT-based planners refined the encodings used to represent bounded plans, for example with propositional encodings [23], abstract CNF encodings [11], and encodings based directly on SAS$^+$ tasks [21]. This direction remains practically relevant, as illustrated by the Madagascar planner [32, 34] and by the broader study of planning-as-satisfiability branching heuristics [33]. Related compilation-based approaches include integer-optimization encodings [25], answer set programming (ASP) [40], and planning-modulo-theories models [16]. Like most of these approaches, our framework solves a sequence of bounded planning problems; unlike them, it targets CP and uses automata together with global constraints as its primary modeling layer.

2.2 Constraint-Based Planning

Constraint-based planning is a special case of compilation-based planning, where bounded planning problems are compiled into constraint satisfaction or optimization models. Early CP approaches focused on parallel plans and used mainly elementary logical constraints [41, 10]. Many later CP models follow the same fixed-horizon pattern, but differ in how they encode transitions and in the degree to which they exploit solver-specific propagation and branching. Zanarini et al. [43] were among the first to model planning objects as finite-state machines and enforce sequential plans with Regular constraints. Later, Barták and Toropila [5], Dvořák et al. [12], and Ghooshchi et al. [14] proposed increasingly compact encodings based on Table constraints.

Other related approaches include constraint-based representations built from composable substate graphs [15], and GP-WCSP, which leverages weighted-CSP models for cost-optimal planning [8]. GP-WCSP compiles planning instances into a weighted-CSP encoding of the planning graph, together with soft constraints that track plan quality and thereby support settings with non-unit action costs.

The closest prior work to ours is that of Babaki et al. [3], who also use automata through Regular and CostRegular [9] constraints, together with counting-based search, but rely on manually derived automata and consequently only report experiments on three planning domains. Our contribution differs in that we derive the automata automatically from standard planning inputs and integrate additional redundant constraints from the planning literature into the resulting CP model.

2.3 Factored Transition Systems and Automata-Based Search

Our work is also closely related to factored state-space representations used in state-space search. Torralba and Sievers [39] show how planning tasks can be reformulated into sets of automata, yielding flexible factored representations that support model reformulation and direct search on the resulting transition systems. We build on the same general idea of representing planning tasks as collections of automata, but use these automata as the basis of a CP encoding rather than as the direct representation on which heuristic search operates. This shift gives access to CP's expressive modeling language, making it comparatively easy to inject global constraints and additional task knowledge into the model.

3 Background

We introduce the planning, automata, and constraint-programming notions used in the paper.

3.1 Classical Planning

Classical planning tasks are typically specified in PDDL [26], the standard input language of the International Planning Competition. Modern planners such as Fast Downward [19] translate PDDL tasks into the ground SAS$^+$ formalism [4] in a preprocessing step. Figure 1 shows an example of a planning task expressed in SAS$^+$.

A SAS$^+$ planning task is a tuple $\Pi = \langle V, A, s_I, G \rangle$, where V is a finite set of *state variables*. Each variable $v \in V$ has a finite domain $dom(v)$. An *atom* is a pair $\langle v, d \rangle$ with $v \in V$ and $d \in dom(v)$. A *partial state* is a function $s : V(s) \to \cup_{v \in V} dom(v)$, where $V(s) \subseteq V$ and $s[v] \in dom(v)$ for all $v \in V(s)$. When $V(s) = V$, we call s a *state*. We often identify partial states with sets of atoms, so that $\langle v, d \rangle \in s$ is equivalent to $s[v] = d$. We write $\mathcal{S}$ for the set

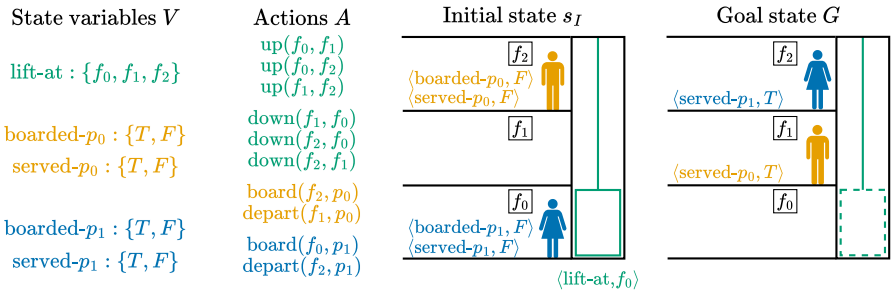

Figure 1 Example SAS$^+$ planning task from the *Miconic* domain, modeling an elevator that serves two passengers on three floors. The task has five state variables, one of which has a domain of size three and the others take Boolean values (T, F). The task defines ten actions. Actions up(f_i, f_j) and down(f_i, f_j) have $\langle$lift-at, $f_i\rangle$ as precondition and $\langle$lift-at, $f_j\rangle$ as effect. Actions board(f_i, p_x) are defined by $\langle\{\langle$lift-at, $f_i\rangle\}, \{\langle$boarded-$p_x, T\rangle\}\rangle$, and depart(f_i, p_x) by $\langle\{\langle$lift-at, $f_i\rangle, \langle$boarded-$p_x, T\rangle, \langle$served-$p_x, F\rangle\}, \{\langle$boarded-$p_x, F\rangle, \langle$served-$p_x, T\rangle\}\rangle$. Note that passengers can only board and depart at their respective starting and destination floors. All other boarding and departing actions are removed during the translation from PDDL to SAS$^+$. Initial state s_I and goal G are given as shown in the figure.

of all states. If r and s are partial states, then updating r with s yields the partial state $t = r \oplus s$ such that $V(t) = V(r) \cup V(s)$, and for every $v \in V(t)$ we have $t[v] = s[v]$ if $v \in V(s)$, and $t[v] = r[v]$ otherwise.

Each action $a \in A$ is a pair $a = \langle pre(a), \mathit{eff}(a)\rangle$, where $pre(a)$ and $\mathit{eff}(a)$ are partial states called the *precondition* and *effect* of a. Action a is *applicable* in a state s if $pre(a) \subseteq s$. Applying a in s yields the state $s[\![a]\!] = s \oplus \mathit{eff}(a)$.

State s_I is the *initial state* and the partial state G is the *goal*. Given a state $s \in \mathcal{S}$, a sequence of actions $\langle a_1, \ldots, a_n\rangle$ is *applicable in* s if there exists a trajectory of states $\langle s_0, s_1, \ldots, s_n\rangle$ such that $s_0 = s$ and, for every $i \in \{1, \ldots, n\}$, action a_i is applicable in s_{i-1} and $s_i = s_{i-1}[\![a_i]\!]$. An *s-plan* for Π is a sequence of actions that is applicable in s and whose final state satisfies the goal, that is, such that $G \subseteq s_n$. When $s = s_I$, an s-plan is simply called a *plan* for Π. Solving Π optimally means finding a shortest plan.

Finally, we use projections of planning tasks onto subsets of variables. For a pattern $P \subseteq V$, the projection of a partial state s onto P is $s_{|P} = \{\langle v, d\rangle \in s \mid v \in P\}$. This extends componentwise to actions, with $a_{|P} = \langle pre(a)_{|P}, \mathit{eff}(a)_{|P}\rangle$, and to planning tasks, with $\Pi_{|P} = \langle P, \{a_{|P} \mid a \in A\}, s_{I|P}, G_{|P}\rangle$.

3.2 Deterministic Finite Automata

An *automaton* is a tuple $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \delta, q_0, F\rangle$, where Σ is a finite *alphabet*, $\mathcal{Q}$ is a finite set of *states*, $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is the *transition function*, $q_0 \in \mathcal{Q}$ is the *initial state*, and $F \subseteq \mathcal{Q}$ is the set of *final states*. A word $w = \langle a_1, \ldots, a_\ell\rangle$ over Σ is *accepted* by $\mathcal{A}$ if there exists a sequence of states $\langle s_0, s_1, \ldots, s_\ell\rangle$ such that $s_0 = q_0$, $s_\ell \in F$, and $\delta(s_{i-1}, a_i) = s_i$ for every $i \in \{1, \ldots, \ell\}$.

Two symbols $a, b \in \Sigma$ are *equivalent* if they induce the same transition from every state, that is, if $\delta(q, a) = \delta(q, b)$ for all $q \in \mathcal{Q}$. We write $a \sim b$ in this case. We also assume that every automaton has a distinguished *infeasible state* $q_i \in \mathcal{Q} \setminus F$ such that $\delta(q_i, a) = q_i$ for every symbol $a \in \Sigma$.

Equivalence of symbols induces a *reduced alphabet* Σ_r together with a surjective *substitution* function $\sigma : \Sigma \to \Sigma_r$ that maps equivalent symbols of Σ to the same reduced symbol. This in turn induces a reduced transition function $\delta_r : \mathcal{Q} \times \Sigma_r \to \mathcal{Q}$ such that $\delta_r(q, \sigma(a)) = \delta(q, a)$ for

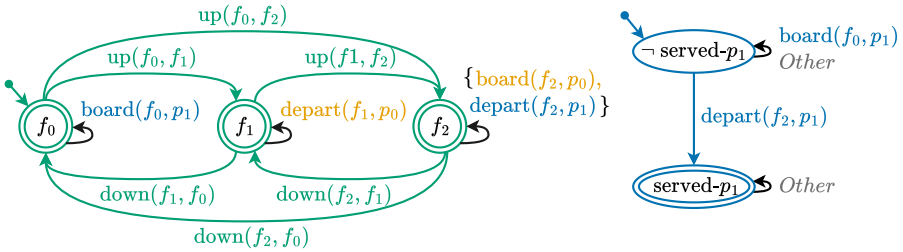

Figure 2 Example automata of two state variables of the SAS$^+$ task from Figure 1. On the left, the automaton $\mathcal{A}^{\text{lift-at}}$ is shown with its reduced alphabet $\Sigma_r^{\text{lift-at}}$ of size $\|\Sigma\| - 1$, where the actions board(f_2, p_0) and depart(f_2, p_1) are mapped to a single symbol. On the right, $\mathcal{A}^{\text{served-}p_1}$ is shown with an alphabet $\Sigma_r^{\text{served-}p_1}$ of size 3, where the only distinct actions of Σ are board(f_0, p_1) and depart(f_2, p_1), while all the remaining actions are mapped to the single symbol *Other*.

every state $q \in \mathcal{Q}$ and symbol $a \in \Sigma$. Since only equivalent symbols are merged, passing to the reduced alphabet preserves language acceptance: a word $w = \langle a_1, \ldots, a_\ell \rangle$ is accepted by $\mathcal{A}$ if and only if $\langle \sigma(a_1), \ldots, \sigma(a_\ell) \rangle$ is accepted by the reduced automaton $\langle \Sigma_r, \mathcal{Q}, \delta_r, q_0, F \rangle$.

3.3 Automata-Based Planning Models

A planning task Π induces a state-space automaton $\mathcal{A}_\Pi$ with alphabet $\Sigma = A$, states $\mathcal{Q} = \mathcal{S} \cup \{q_i\}$, initial state $q_0 = s_I$, and final states $F = \{s \in \mathcal{S} \mid G \subseteq s\}$. Its transition function satisfies $\delta(s, a) = s[\![a]\!]$ if a is applicable in s, and $\delta(s, a) = q_i$ otherwise, for every state $s \in \mathcal{S}$. In the automata setting, we say that an action is *applicable* in a state whenever its transition does not lead to the infeasible state.

A *factored representation* of the state space of Π is a set of automata $S = \{\mathcal{A}^1, \ldots, \mathcal{A}^n\}$ over the common alphabet A. The representation is *exact* if a sequence of actions $p = \langle a_1, \ldots, a_\ell \rangle$ is a plan for Π if and only if p is accepted by every automaton in S, and the product of these automata yields the full state-space automaton $\mathcal{A}_\Pi$ [37].

The *atomic representation* of Π is the exact factored representation $S = \{\mathcal{A}^{v_1}, \ldots, \mathcal{A}^{v_{|V|}}\}$, where each automaton $\mathcal{A}^{v_i}$ is induced by the projected planning task $\Pi_{|\{v_i\}}$. Figure 2 shows two such automata for the example task of Figure 1.

3.4 Constraint Programming

A constraint satisfaction problem (CSP) is a triple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X}$ is a finite set of variables, $\mathcal{D}$ is a finite set of values, and each variable $x \in \mathcal{X}$ has a finite domain $\mathcal{D}(x) \subseteq \mathcal{D}$. Each constraint $c \in \mathcal{C}$ is a relation on a subset of the variables in $\mathcal{X}$. Solving a CSP amounts to finding an assignment of the variables in $\mathcal{X}$ that satisfies all constraints simultaneously.

We next describe the constraints used in our CP model. Regular$(\langle x_1, x_2, \ldots, x_n \rangle, \mathcal{A})$ holds if the values taken by the sequence of finite-domain variables $\langle x_1, x_2, \ldots, x_n \rangle$ form a word in the regular language recognized by $\mathcal{A}$. Table$(\langle x_1, x_2, \ldots, x_n \rangle, \mathcal{T})$ is satisfied if the tuple of values taken by the variables belongs to the relation $\mathcal{T}$ given in extension. Among$(o, \{x_1, x_2, \ldots, x_n\}, \mathcal{V})$ enforces that variable o is equal to the number of variables in $\{x_1, x_2, \ldots, x_n\}$ taking on a value belonging to $\mathcal{V}$. Sum$(s, \{x_1, x_2, \ldots, x_n\})$ enforces that $s = x_1 + x_2 + \cdots + x_n$.

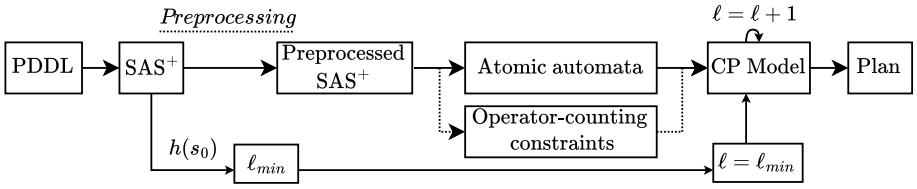

Figure 3 Pipeline of our approach. The planning task is first translated from PDDL to SAS$^+$, and then preprocessed [2]. Redundant constraints are extracted from this representation, which is then translated into a set of automata, one for each state variable. Next, we build a CP model that is solved iteratively, starting from a lower bound extracted from the SAS$^+$ model, until a plan is found. Optional components of the pipeline are shown with dotted lines and are enabled or disabled in our experiments to evaluate their impact.

4 Planning as CP with Automata

Our approach reformulates a planning task as a collection of automata that capture its dynamics and then compiles this representation into a CP model. Starting from a planning problem specified in PDDL, we automatically generate the automata, post the corresponding REGULAR constraints, and solve the resulting model. This section first describes the overall pipeline and then presents our base CP model, inspired by the encoding of Babaki et al. [3].

4.1 Algorithm Outline

Figure 3 summarizes the main stages of our approach. We begin by translating the input task from first-order PDDL into a ground finite-domain SAS$^+$ task. From this representation, we compute an admissible lower bound $\ell_{\min}$ on the optimal plan length by evaluating an admissible heuristic in the initial state.[1]

The SAS$^+$ task is then preprocessed using an off-the-shelf tool [2]. Afterwards, we convert the preprocessed task into its atomic factored representation, that is, the set of automata $S = \{\mathcal{A}^{v_1}, \ldots, \mathcal{A}^{v_{|V|}}\}$ introduced in Section 3.3. From the same representation, we also derive the operator-counting constraints discussed in Section 5, which capture global properties of the task and are added to the model as redundant constraints.

Finally, the automata and the additional constraints are compiled into a CP model for a fixed plan horizon ℓ. We solve these models in an iterative-deepening scheme, starting from $\ell_{\min}$ and increasing ℓ by one until a solution is found. Since every model with horizon smaller than $\ell_{\min}$ is infeasible and horizons are considered in increasing order, the first solution found has the optimal plan length.

4.2 Preprocessing

The preprocessing phase modifies the SAS$^+$ task before the automata are generated. It uses an off-the-shelf preprocessor [2] to prune irrelevant atoms and actions, and it can also enrich the task with implied atoms in action preconditions and in the goal description. Although preprocessing does not change the set of state variables and therefore does not affect the number of automata, it can make the resulting automata more informative.

[1] We compute the lower bound $\ell_{\min}$ on the initial model, before preprocessing. This ensures a fair comparison between preprocessing configurations, since the heuristic value may otherwise change because of the preprocessing itself.

The first enrichment adds implied atoms to action preconditions. These *implied preconditions* remove transitions that are actually inapplicable, making explicit that some actions can never be taken from certain partial states in the projected automata. The second enrichment adds implied atoms to the goal. These *implied goals* reduce the set of accepting states.[2]

In our experiments, pruning irrelevant atoms and actions is always enabled since this can only help the CP solver. In contrast, the two implied-atom enrichments can have more variable effects on CP performance, so we evaluate them separately and jointly.

4.3 Base CP Model

Our base CP model encodes the bounded planning problem for a fixed horizon ℓ. Following Babaki et al. [3], it uses Regular constraints to enforce the dynamics of the automata in the atomic representation. For each $k \in \{1, \dots, n\}$, we define the *reduced* automaton $\mathcal{A}^k = \langle \Sigma_r^k, \mathcal{Q}^k, \delta^k, q_0^k, F^k \rangle$. Each automaton has a reduced alphabet Σ_r^k, together with a substitution function σ^k that maps equivalent actions of A to symbols of Σ_r^k. The model uses the following variables:

- A sequence $plan = \langle x_1, \dots, x_\ell \rangle$, such that $\mathcal{D}(x_i) \subseteq A$;
- For each automaton $\mathcal{A}^k$, a sequence $plan^k = \langle x_1^k, \dots, x_\ell^k \rangle$ such that $\mathcal{D}(x_i^k) \subseteq \Sigma_r^k$.

Over these variables, the model contains two kinds of constraints. First, for each automaton $\mathcal{A}^k$, we post the following Regular constraint over the sequence $plan^k$:

$$\textsc{Regular}(\langle x_1^k, \dots, x_\ell^k \rangle, \mathcal{A}^k) \quad \forall k,\ 1 \le k \le n$$

We also link each action variable x_i^k to the corresponding global variable x_i according to Σ_r^k, through the following Table constraints,

$$\textsc{Table}(\langle x_i, x_i^k \rangle, \mathcal{T}^k) \quad \forall k,\ 1 \le k \le n,\ \forall i,\ 1 \le i \le \ell,$$

where $\mathcal{T}^k = \{\langle a, \sigma^k(a) \rangle \mid a \in A\}$. Note that, in the special case where $\Sigma_r^k = A$ and σ^k is the identity function, the Table constraints are not necessary, and we directly post the Regular constraint on the global variables of *plan*. Figure 4 shows, through an example, the interactions between the variables and the constraints that we define in our base model.

The main benefit of using the reduced alphabets Σ_r^k is that the associated Regular constraints become cheaper to propagate. This comes at the cost of introducing the auxiliary variables $plan^k$ and the linking Table constraints, but preliminary experiments showed that the trade-off is often beneficial. The reason is that each automaton captures only a restricted aspect of the planning task, so many actions have identical behaviour in that automaton and can be merged into a single reduced symbol.

5 Counting Actions

Factored representations decompose the enormous state space of a planning task into a collection of compact automata, which pair well with Regular constraints and can therefore be propagated efficiently. However, this factoring also localizes information that could enable additional pruning if combined across automata. In particular, some interactions between actions become visible only when several variables are considered jointly.

[2] The implied atoms are inferred by a forward analysis of the relaxed state space that considers pairs of atoms to compute mutexes (mutually exclusive atoms). Action preconditions are then *disambiguated* by solving a CSP that enforces consistency between the values taken by atoms in the preconditions and the mutex information [1].

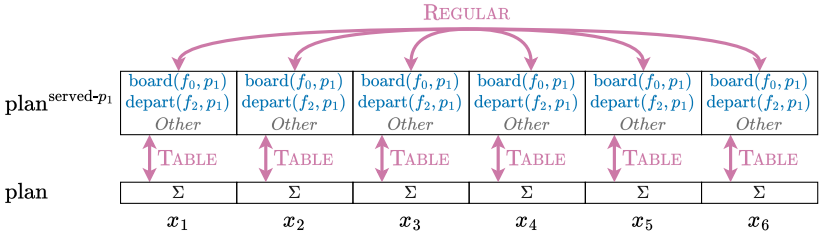

Figure 4 Constraints added for the automaton $\mathcal{A}^{\text{served-}p_1}$ of Figure 2 for a plan of length $\ell = 6$. The domains of the variables $x_i^{\text{served-}p_1}$ are reduced to the symbols of $\Sigma_r^{\text{served-}p_1}$, and the Regular constraint is posted on these variables. The Table constraints link each variable $x_i^{\text{served-}p_1}$ to the corresponding global variable x_i according to the mapping given by $\sigma^{\text{served-}p_1}$.

In the planning literature, such global information is often inferred by analyzing the planning task in its SAS$^+$ representation. Here, we focus on *operator-counting constraints* [30], that is, linear inequalities over the number of occurrences α_a of action a in a candidate plan.

These constraints strengthen our Base model with information that would otherwise remain implicit in the individual automata, allowing the solver to prune the search space more effectively. We consider two types of operator-counting constraints: *landmark constraints* [22, 7] and *net change constraints* [6]. We first present these constraints in their standard form, and then show how to compile them more compactly by grouping action counts.

5.1 Landmark Constraints

A *disjunctive action landmark* [20] is a set $L \subseteq A$ such that at least one action in L must appear in every plan. Deciding whether a set L of actions is a landmark is as hard as solving the planning problem itself [31], and planners therefore identify landmarks by analyzing relaxations of the task. We use such an off-the-shelf tool to compute landmarks [20], and each landmark $L \subseteq A$ is included in our model through the following *landmark constraint*:

$$\sum_{a \in L} \alpha_a \geq 1$$

5.2 Net Change Constraints

Net change constraints generalize the following simple observation: if an atom $\langle v, d \rangle$ belongs to the goal but not to the initial state, then any plan must establish it one more time than it deletes it. We present only the subset of net change constraints needed in our model and refer to the literature for a more exhaustive description [30]. Formally, the net change of an atom $\langle v, d \rangle$ is $\delta_{\langle v,d \rangle}$, which represents, for any plan, the difference between the number of times the atom must be established and the number of times it must be deleted.

$$\delta_{\langle v,d \rangle} = \begin{cases} 1 & \text{if } G[v] = d \text{ and } s_I[v] \neq d \\ -1 & \text{if } G[v] \neq d \text{ and } s_I[v] = d \\ 0 & \text{otherwise.} \end{cases}$$

Some actions always establish an atom, i.e., make that atom true while it was false before the application of the action. These actions *surely add* atom $\langle v, d \rangle$:

$$SA_{\langle v,d \rangle} = \{a \in A \mid \mathit{eff}(a)[v] = d \text{ and } \mathit{pre}(a)[v] \neq d\}$$

Similarly, the following sets of actions respectively *possibly add* and *surely delete* atom $\langle v, d\rangle$:

$$PA_{\langle v,d\rangle} = \{a \in A \mid \mathit{eff}(a)[v] = d \text{ and } \mathit{pre}(a)[v] \text{ undefined}\}$$
$$SD_{\langle v,d\rangle} = \{a \in A \mid \mathit{eff}(a)[v] \neq d \text{ and } \mathit{pre}(a)[v] = d\}$$

We then obtain constraints of the following form:

$$\sum_{a \in SA_{\langle v,d\rangle}} \alpha_a + \sum_{a \in PA_{\langle v,d\rangle}} \alpha_a - \sum_{a \in SD_{\langle v,d\rangle}} \alpha_a \geq \delta_{\langle v,d\rangle}$$

The left-hand side of the inequality represents an upper bound on the difference between the number of times an atom is established and the number of times it is deleted. This quantity is therefore itself lower-bounded by the net change of the atom.

5.3 Grouping Action Occurrences

The two families of constraints above are particularly convenient because all coefficients of action counts α_a belong to $\{-1, 0, 1\}$. This property lets us track action counts jointly across operator-counting constraints using Among constraints. It therefore factors the information into a connected network of variables and constraints that gives the solver more opportunities to prune the search space.

To this end, we first determine a suitable partition $\mathcal{P}$ of the action set, which we do automatically from the operator-counting constraints. Intuitively, two actions a and b can be grouped together if their counts α_a and α_b always appear with the same coefficient in every constraint under consideration. We start from the coarsest partition $\mathcal{P} = \{A\}$ and refine it sequentially over the constraints, as described in Algorithm 1. Whenever a set contains actions that occur with different coefficients in the current constraint, we split that set accordingly. Actions absent from a constraint are treated as having coefficient 0. The resulting partition is given by $\mathcal{P} = \{D_1, \ldots, D_m\}$ such that $\bigsqcup_{i \leq m} D_i = A$, where each D_i is a disjoint subset of actions that can be counted together.

Algorithm 1 Partition refinement for grouping action counts.

```
P ← {A}
for all constraints c ∈ C in the model do
    P' ← ∅
    for all sets D ∈ P do
        for all q ∈ {−1, 0, 1} do
            D_q ← {a ∈ D | q is the coefficient of α_a in c}
            if D_q ≠ ∅ then
                P' ← P' ∪ {D_q}
    P ← P'
return P
```

For each set D_i, we introduce a variable ϑ_i with domain $\mathcal{D}(\vartheta_i) = \{0, \ldots, \ell\}$, where ℓ is the plan length. It is linked to the plan through the constraint

$$\textsc{Among}(\vartheta_i, \{x_1, x_2, \ldots, x_\ell\}, D_i) \quad \forall i,\ 1 \leq i \leq m.$$

We also add the following constraint to ensure that the total number of occurrences equals ℓ:

$$\textsc{Sum}(\ell, \{\vartheta_1, \vartheta_2, \ldots, \vartheta_m\})$$

We now show how landmark and net change constraints are compiled once such a partition has been computed.

5.3.1 Grouped Landmark Constraints

Observe that, after partition refinement, we obtain a partition $\mathcal{P}$ in which each set contains actions that either always belong to the same disjunctive landmark or never appear in it. For each disjunctive action landmark L, we introduce an associated occurrence variable ϑ^L with domain $\{1, \dots, \ell\}$, and the constraint is posted as

$$\textsc{Sum}(\vartheta^L, \bigsqcup_{\substack{1 \leq i \leq m \text{ s.t.} \\ D_i \subseteq L}} \{\vartheta_i\}). \tag{1}$$

▶ **Example 1.** Suppose that we have the landmarks $L_1 = \{a_1, a_2, a_3, a_4\}$, $L_2 = \{a_2, a_3\}$, and $L_3 = \{a_1, a_4, a_5\}$. For notational convenience, we associate action count α_i to action a_i, for all $i \in \{1, \dots, 5\}$, and obtain the following landmark constraints:

$$\begin{aligned} \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 &\geq 1 \\ \alpha_2 + \alpha_3 &\geq 1 \\ \alpha_1 + \alpha_4 + \alpha_5 &\geq 1 \end{aligned}$$

This system induces the partition $\mathcal{P} = \{\{a_1, a_4\}, \{a_2, a_3\}, \{a_5\}\}$, with associated variables $\vartheta_{1,4}$, $\vartheta_{2,3}$, and ϑ_5, in respective order. The left-hand sides of the equations are then respectively replaced by variables ϑ^{L_1}, ϑ^{L_2}, and ϑ^{L_3}, which must satisfy the following constraints:

$$\begin{aligned} &\textsc{Sum}(\vartheta^{L_1}, \{\vartheta_{1,4}, \vartheta_{2,3}\}) \\ &\textsc{Sum}(\vartheta^{L_2}, \{\vartheta_{2,3}\}) \\ &\textsc{Sum}(\vartheta^{L_3}, \{\vartheta_{1,4}, \vartheta_5\}) \end{aligned}$$

5.3.2 Grouped Net Change Constraints

The partitions induced by these constraints must account for the fact that not all action occurrences appear with positive coefficients. This is why our partition-refinement procedure yields two disjoint sets of occurrences: one for positive coefficients, λ_p, and one for negative coefficients, λ_n. After refinement, for each constraint we introduce variables ϑ^{λ_p} and ϑ^{λ_n} with domain $\{0, \dots, \ell\}$ representing the corresponding sums, defined as follows for $\lambda \in \{\lambda_p, \lambda_n\}$:

$$\textsc{Sum}(\vartheta^{\lambda}, \bigsqcup_{\substack{1 \leq i \leq m \text{ s.t.} \\ D_i \subseteq \lambda}} \{\vartheta_i\})$$

We then post the grouped net change constraint. For an atom $\langle v, d \rangle$ with net change $\delta_{\langle v,d \rangle} \in \{-1, 0, 1\}$, the inequality $\vartheta^{\lambda_p} - \vartheta^{\lambda_n} \geq \delta_{\langle v,d \rangle}$ is encoded as

$$\textsc{GreaterOrEqual}(\vartheta^{\lambda_p}, \vartheta^{\lambda_n} + \delta_{\langle v,d \rangle}). \tag{2}$$

6 Experiment Setup

Our experimental pipeline mirrors the one described in Section 4.1. Starting from a planning task in PDDL, we use Scorpion [36], an extension of the Fast Downward planning system [19], to generate a finite-domain SAS$^+$ task. On this initial task, we compute the lower bound $\ell_{\min}$ on plan length as the value of the admissible LM-Cut heuristic [20] in the initial state. We then pass the task to Scorpion again for preprocessing [2], adding implied atoms. Depending on the configuration, preprocessing additionally introduces implied preconditions ($-^{\text{IP}}$), implied goals ($-^{\text{IG}}$), or both ($-^{\text{IPG}}$). Finally, we call Scorpion once more to compute the atomic representation and, when needed, the operator-counting constraints.

Table 1 Comparison of different preprocessing techniques. We compare our BASE model to models enriched with implied preconditions ($-^{\text{IP}}$), implied goals ($-^{\text{IG}}$), or both ($-^{\text{IPG}}$). The first four columns compare the different preprocessing techniques on a per-domain basis, where each cell holds the number of domains for which the approach in the row performed better than the approach in the column. The last column reports the number of tasks solved.

	BASE	BASE$^{\text{IP}}$	BASE$^{\text{IG}}$	BASE$^{\text{IPG}}$	Total (1786)
BASE	–	2	5	0	675
BASE$^{\text{IP}}$	**2**	–	6	0	676
BASE$^{\text{IG}}$	**6**	**7**	–	1	683
BASE$^{\text{IPG}}$	**7**	**7**	**7**	–	**690**

For a fixed plan length ℓ, we build the model BASE from Section 4.3 using the CP-SAT solver [27]. When available, we also add the landmark constraints (LMC) from Equation 1 or the net change constraints (NCC) from Equation 2. Starting with $\ell = \ell_{\min}$, we iteratively solve the model for increasing values of ℓ until a solution is found.

We evaluate our approach on benchmarks from the Optimal Tracks of the International Planning Competition (IPC) 1998–2023, modified to use unit action costs. Our benchmark set is composed of 1786 instances, divided into 47 planning *domains*. All experiments are run on Intel Xeon Gold 6130 processors with a time limit of 30 minutes and a memory limit of 8 GiB per run. Our implementation is available online [42].

We compare our models against three types of baselines. As a CP-based baseline, we use GP-WCSP [8], a weighted-CSP approach that is, to our knowledge, the state of the art among CP-based optimal classical planners. We adapted it to use CP-SAT as the solver, ensuring a fair comparison. We also compare against the automata-based CP approach of Babaki et al. [3]. To isolate the effect of the automata themselves, we evaluate their approach by replacing the automata in our base model with their manually designed ones.

Finally, we include the heuristic-search baselines used in the latest IPC, namely blind search (BLIND) and A* with the LM-Cut heuristic (LM-CUT) [20]. The latter represents a strong optimal-planning baseline.

7 Experiment Results

In this section, we present the results of our experimental evaluation. We first examine the effect of task preprocessing by comparing our different configurations. We then evaluate the impact of the redundant constraints and, finally, compare our strongest models with the state of the art in CP-based and heuristic-search planning.

7.1 Impact of Task Preprocessing

We begin by evaluating the effect of preprocessing on the base model BASE. We consider three variants: BASE$^{\text{IP}}$ with implied preconditions, BASE$^{\text{IG}}$ with implied goals, and BASE$^{\text{IPG}}$ with both. Table 1 shows that preprocessing has a measurable but uneven impact. Across the 47 domains, the preprocessing step often identifies additional preconditions and goals and therefore yields more informative automata, but this does not always translate into a higher number of solved instances. When applied individually, implied preconditions and implied goals each increase the total number of solved tasks, yet both yield mixed results

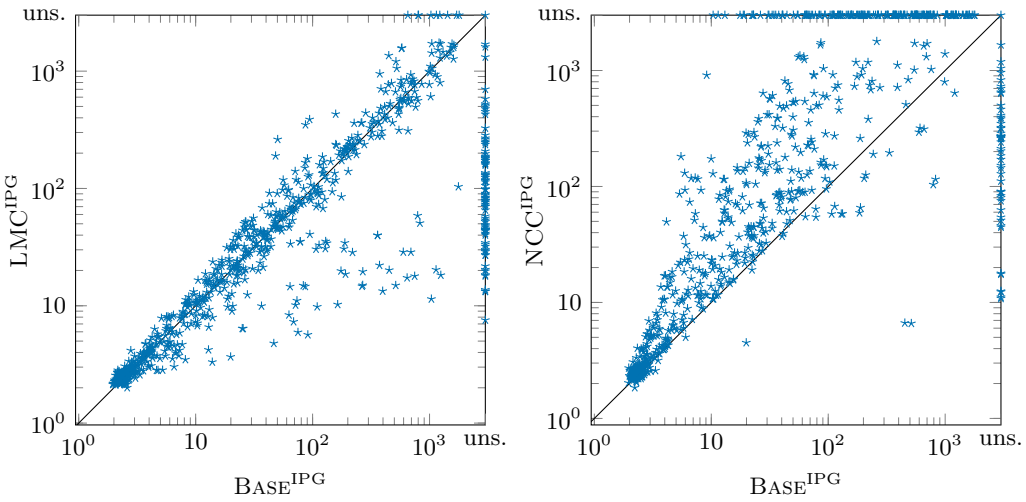

Figure 5 Comparison of overall runtime for BASE^{IPG} against LMC^{IPG} and NCC^{IPG}. BASE^{IPG} is our base model, LMC^{IPG} is our model with landmark constraints, and NCC^{IPG} is our model with net change constraints. All models are enriched with implied preconditions and goals. Points on the "uns." axes indicate runs that hit the time or memory limit.

at the per-domain level. The combined variant is clearly the strongest: BASE^{IPG} has the largest overall number of solved tasks and outperforms the base model in 7 domains, while the base model never outperforms it on any domain. These 7 domains are precisely the union of the domains on which BASE^{IP} and BASE^{IG} outperform BASE individually. This suggests that the two preprocessing techniques are complementary and that combining them yields a strictly more informative model.

7.2 Impact of Redundant Constraints

We now take the strongest preprocessed model, BASE^{IPG}, and study the effect of adding the two families of redundant constraints introduced earlier: the landmark constraints LMC^{IPG} and the net change constraints NCC^{IPG}.

The landmark constraints are the most successful extension overall. As shown in Table 2, LMC^{IPG} improves the total number of solved tasks over BASE^{IPG} and is particularly effective in domains such as *Logistics*, *Miconic*, *Satellite*, *Scanalyzer*, *Woodworking* and *Zenotravel*. Figure 5 shows that these additional constraints come at a cost, as they increase runtime for some instances. Overall, however, the extra information they provide is often beneficial enough to compensate for that overhead and to solve tasks that remain out of reach for the base model. The domains for which these constraints are most effective share properties that make the constraints particularly informative for our model. One of these properties is that the constraints often contain many disjunctive action landmarks whose action sets overlap little. In that case, the occurrence variables ϑ_i get tighter domains, which rules out candidate actions earlier in the search. If the number of such constraints reaches the length of the optimal plan, which is often the case in the *Miconic* domain, then they nearly determine the actions that must occur in the plan and leave only their ordering to be decided by the solver, thereby greatly reducing the search space.

The picture is different for the net change constraints. Table 2 shows that NCC^{IPG} performs substantially worse overall than both BASE^{IPG} and LMC^{IPG}. This is consistent with Figure 5, where the corresponding runs are typically slower. One possible explanation

Table 2 Number of solved tasks per domain for BASE^{IPG} and its two extensions with additional operator-counting constraints, LMC^{IPG} and NCC^{IPG}. Domain sizes (number of tasks) are shown in parentheses. A domain is detailed in the table if the absolute difference between LMC^{IPG} and BASE^{IPG} is at least 2 or if the difference between NCC^{IPG} and BASE^{IPG} is more than 2. The remaining domains are compiled in the *Other* column.

	DataNetwork (20)	Logistics (63)	Miconic (150)	ParcPrinter (50)	Pegsol (50)	Rovers (40)	Satellite (36)	Scanalyzer (50)	Tetris (17)	Trucks (30)	Woodworking (50)	Zenotravel (20)	*Other* (1210)	Total (1786)
BASE^{IPG}	**17**	13	58	30	38	4	7	23	5	5	35	9	446	690
LMC^{IPG}	15	**28**	**102**	34	38	**6**	**9**	**25**	5	5	**50**	**11**	443	**771**
NCC^{IPG}	12	13	38	**50**	**42**	5	5	11	**9**	**8**	45	8	323	569

is that these constraints induce a finer partition of the action set, which increases the number of occurrence variables ϑ_i and constraints, raising both the runtime and the memory consumption of the model. Indeed, out of the 1786 benchmark tasks, 876 reached the memory limit for NCC^{IPG}, compared to 454 for BASE^{IPG}. Of these, 518 and 85 cases, respectively, occurred during model construction, while the remaining cases occurred during solving. For comparison, LMC^{IPG} had 524 out-of-memory runs, including 98 during model construction.

Despite this overall drop, the net change constraints are not uniformly ineffective. They outperform both BASE^{IPG} and LMC^{IPG} on four domains, as shown in Table 2. These domains are characterized by actions that can be grouped into small clusters within which actions are highly interdependent and must be applied in a certain order. This appears to be especially true in the *ParcPrinter* domain, where the NCC constraints induce cascading requirements on which actions must occur. Propagating these requirements can greatly reduce the search space.

7.3 Comparison with Baselines

We now compare our strongest configurations with the baselines. Table 3 shows that the best-performing version of our approach, LMC^{IPG}, clearly improves over the state-of-the-art CP-based baseline GP-WCSP. This confirms the effectiveness of automata-based CP encodings. That said, the comparison should be interpreted with some care because GP-WCSP was designed primarily for cost-optimal planning with arbitrary action costs, whereas our experiments focus on unit-cost optimal planning.

As a second CP-based reference point, we compare against the manually designed automata of Babaki et al. [3], who consider three domains: *Floortile*, *Miconic* and *Scanalyzer*. We evaluate their automata definitions on an instance set of 150 *Miconic*, 41 *Scanalyzer*, and 34 *Floortile* tasks. Substituting these automata into our pipeline and running it on these instances, leads to solving 51 *Miconic*, 21 *Scanalyzer*, and 6 *Floortile* instances, while LMC^{IPG} solves 102, 25, and 5, respectively. This comparison should again be read carefully: the solver used by Babaki et al. relies on branching heuristics that explicitly exploit the structure of their automata, whereas we use CP-SAT with its default search. Still, the results suggest that manual automaton design alone is not enough. To be effective, it likely needs to be paired with a search strategy that can exploit the resulting higher-level structure.

Table 3 Per-domain comparison of our models (BASE^{IPG} and LMC^{IPG}) and the baselines. Each cell holds the number of domains for which the approach in the row solved more tasks than the approach in the column. Bold numbers indicate that the approach in the row outperformed the one in the column on more domains than the reverse comparison.

	GP-WCSP	BASE^{IPG}	LMC^{IPG}	BLIND	LM-CUT	Total (1786)
GP-WCSP	–	3	2	3	1	401
BASE^{IPG}	**39**	–	7	13	7	690
LMC^{IPG}	**40**	**11**	–	15	10	771
BLIND	**41**	**26**	**23**	–	15	784
LM-CUT	**44**	**32**	**30**	**26**	–	**965**

Our approach does not yet match the overall performance of the heuristic-search baselines BLIND and LM-CUT. Nevertheless, it remains competitive on a meaningful subset of domains, which points to complementary strengths rather than outright dominance. Figure 6 reinforces this view: while the heuristic-search baselines retain the lead, the cumulative coverage curves of our strongest configurations continue to rise even close to the time limit, suggesting that part of the remaining gap is due to runtime rather than to a lack of pruning power. More generally, these results make portfolio combinations of CP-based and heuristic-search techniques a natural direction for future work.

While both LM-CUT and LMC^{IPG} rely heavily on landmark information, LM-CUT performs substantially better overall, solving 965 tasks compared to 771 for our approach. Still, LMC^{IPG} solves more instances in 10 of the 47 domains. One notable example is *Woodworking*, where our approach solves all 50 instances within a few seconds, while LM-CUT solves only 39. This advantage appears to stem from a favourable combination of factors. First, the lower bound $\ell_{\min}$ that we compute is often very close to the optimal plan length ℓ^*, so our algorithm has to prove infeasibility for only a few horizons below ℓ^*. Second, the landmark constraints identify many actions that must occur in a valid plan. Finally, a *Woodworking* instance often decomposes into many smaller and relatively independent subproblems, which makes the ordering between actions highly flexible. This turns the domain into a loosely constrained scheduling problem, a setting in which CP is often particularly effective. A similar phenomenon occurs in the *Visitall* domain, where LMC^{IPG} solves 22 instances and LM-CUT only 16.

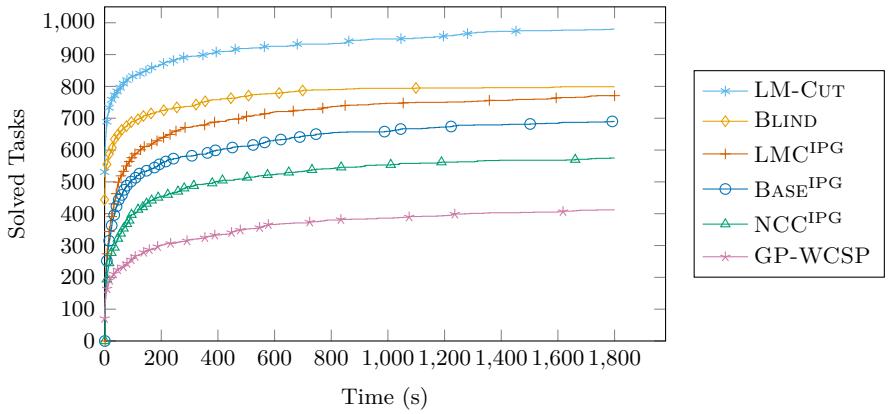

Figure 6 Accumulated number of solved tasks over time for our models ($-^{IPG}$) and the baselines.

8 Discussion

Our results show that automatically generated automata can support competitive CP models for optimal planning, especially when combined with suitable redundant constraints. At the same time, they also highlight the current limits of the approach: some classes of constraints help only on specific domains, the present search procedure is tailored to unit-cost planning, and the automata representation remains less expressive than richer factored planning formalisms. We discuss these three directions below.

8.1 Additional Constraints

The results of Section 7 suggest that redundant constraints should not be treated as universally beneficial add-ons. Landmark constraints improve performance overall, whereas net change constraints help only on a smaller subset of domains and can substantially increase the size of the model. This points to a more selective use of redundant constraints, where the modeling pipeline decides automatically which families of constraints are worth instantiating for a given task.

One possible direction is to predict the usefulness of a constraint family from lightweight structural features of the compiled task, such as the size of the automata, the amount of overlap between landmark action sets, the granularity of the action partition induced by the constraints, or the expected increase in variables and propagators. Another possibility is to use a short probing phase: one could build several candidate models, solve them only briefly, and then continue with the variant that exhibits the best early progress.

More generally, the planning literature offers additional types of operator-counting constraints [30]. For example, post-hoc optimization constraints [29] use an admissible heuristic estimate together with action costs to prune candidate plans whose total cost would violate the heuristic lower bound. Because such constraints depend closely on the chosen heuristic, we leave their integration into our framework for future work.

8.2 Extension to General Costs

Our current solution procedure targets unit-cost planning and can therefore conduct search by increasing plan length. Extending the approach to general action costs would require reasoning about plan cost directly rather than using horizon alone as the optimization criterion. At a high level, the existing compilation pipeline could be preserved, but the resulting CP model would need to incorporate a cost variable or cost-aware global constraints such as CostRegular. This suggests two natural solution strategies. One is to keep the current iterative scheme and replace horizon bounds with admissible cost bounds. The other is to build a single optimization model and let the solver minimize total cost directly. Which option is preferable depends on how well the CP solver can exploit cost structure in the presence of the automata constraints.

8.3 Automata Reformulations

An interesting direction for extending our framework is to support richer factored representations and automaton reformulations. Many transformations from the planning literature, such as merge-based methods [38], construct larger automata through non-deterministic intermediate objects. This does not fit naturally with our current encoding, which is designed for a factored collection of deterministic automata enforced through Regular constraints. Supporting such transformations would therefore require either richer CP constraints that

can reason over non-deterministic transitions or new compilation techniques that preserve the benefits of factored determinism while gaining some of the expressive power of richer factored representations.

9 Conclusions

We presented an end-to-end, automated CP framework for optimal classical planning. Starting from a PDDL task, our pipeline translates it into SAS$^+$, computes an atomic factored representation as a set of DFAs, and constructs a CP model based on Regular constraints that is solved by iterative deepening over the plan horizon. Unlike prior automata-based CP approaches, no manual engineering of automata is required.

We also showed that the framework is easily extensible: redundant operator-counting constraints derived from the planning literature, such as landmark constraints and net change constraints, can be integrated directly into the CP model and improve search performance in a number of domains. Extensive experiments show that our best configuration outperforms the state-of-the-art CP-based planner, is competitive with blind heuristic-search planners, and establishes end-to-end automata-based CP as a viable direction for optimal planning.

References

1 Vidal Alcázar, Daniel Borrajo, Susana Fernández, and Raquel Fuentetaja. Revisiting regression in planning. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 2254–2260. AAAI Press, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6877.

2 Vidal Alcázar and Álvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Proc. ICAPS 2015*, pages 2–6, 2015. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10518.

3 Behrouz Babaki, Gilles Pesant, and Claude-Guy Quimper. Solving classical AI planning problems using planning-independent CP modeling and search. In *Proc. SoCS 2020*, pages 2–10, 2020. doi:10.1609/SOCS.V11I1.18529.

4 Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995. doi:10.1111/J.1467-8640.1995.TB00052.X.

5 Roman Barták and Daniel Toropila. Reformulating constraint models for classical planning. In *Proc. FLAIRS 2008*, pages 525–530, 2008. URL: http://www.aaai.org/Library/FLAIRS/2008/flairs08-121.php.

6 Blai Bonet. An admissible heuristic for SAS$^+$ planning obtained from the state equation. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 2268–2274. AAAI Press, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6927.

7 Blai Bonet and Malte Helmert. Strengthening landmark heuristics via hitting sets. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 329–334. IOS Press, 2010. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=17767.

8 Martin C. Cooper, Marie de Roquemaurel, and Pierre Régnier. A weighted CSP approach to cost-optimal planning. *AI Communications*, 24(1):1–29, 2011. doi:10.3233/AIC-2010-0473.

9 Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006. doi:10.1007/S10601-006-9003-7.

10 Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001. doi:10.1016/S0004-3702(01)00128-X.

11 Carmel Domshlak, Jörg Hoffmann, and Ashish Sabharwal. Friends or foes? On planning as satisfiability and abstract CNF encodings. *Journal of Artificial Intelligence Research*, 36:415–469, 2009. doi:10.1613/JAIR.2817.

12 Filip Dvořák, Daniel Toropila, and Roman Barták. Yet more planning efficiency: Finite-domain state-variable reformulation. *Journal of Experimental & Theoretical Artificial Intelligence*, 27(5):543–576, 2015. doi:10.1080/0952813X.2014.993504.

13 Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

14 Nina Ghanbari Ghooshchi, Majid Namazi, M.A.Hakim Newton, and Abdul Sattar. Encoding domain transitions for constraint-based planning. *Journal of Artificial Intelligence Research*, 58:905–966, 2017. doi:10.1613/JAIR.5378.

15 Peter Gregory, Derek Long, and Maria Fox. Constraint based planning with composable substate graphs. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 453–458. IOS Press, 2010. doi:10.3233/978-1-60750-606-5-453.

16 Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. Planning modulo theories: Extending the planning paradigm. In *Proc. ICAPS 2012*, pages 65–73, 2012.

17 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

18 Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*, volume 13 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, 2019.

19 Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. doi:10.1613/JAIR.1705.

20 Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, pages 162–169, 2009.

21 Ruoyun Huang, Yixin Chen, and Weixiong Zhang. SAS^+ planning as satisfiability. *Journal of Artificial Intelligence Research*, 43:293–328, 2012. doi:10.1613/JAIR.3442.

22 Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *Proc. IJCAI 2009*, pages 1728–1733, 2009. URL: http://ijcai.org/Proceedings/09/Papers/288.pdf.

23 Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proc. KR 1996*, pages 374–384, 1996.

24 Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI 1992*, pages 359–363, 1992.

25 Henry A. Kautz and Joachim P. Walser. State-space planning by integer optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI 1999)*, pages 526–533. AAAI Press, 1999. URL: http://www.aaai.org/Library/AAAI/1999/aaai99-075.php.

26 Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University, 1998.

27 Laurent Perron and Frédéric Didier. CP-SAT. https://developers.google.com/optimization/cp/cp_solver/, 2025. Version v9.12, Google.

28 Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *Proc. CP 2004*, pages 482–495, 2004. doi:10.1007/978-3-540-30201-8_36.

29 Florian Pommerening, Gabriele Röger, and Malte Helmert. Getting the most out of pattern databases for classical planning. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 2357–2364. AAAI Press, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6965.

30 Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. LP-based heuristics for cost-optimal planning. In *Proc. ICAPS 2014*, pages 226–234, 2014.

31 Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proc. ECP 2001*, pages 174–182, 2001.

32 Jussi Rintanen. Madagascar: Scalable planning with SAT. In *IPC 2011 Planner Abstracts*, pages 61–64, 2011.

33 Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012. doi:10.1016/J.ARTINT.2012.08.001.

34 Jussi Rintanen. Madagascar: Scalable planning with SAT. In *IPC-8 Planner Abstracts*, pages 66–70, 2014.

35 Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

36 Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67:129–167, 2020. doi:10.1613/JAIR.1.11673.

37 Silvan Sievers and Malte Helmert. Merge-and-shrink: A compositional theory of transformations of factored transition systems. *Journal of Artificial Intelligence Research*, 71:781–883, 2021. doi:10.1613/JAIR.1.12557.

38 Silvan Sievers, Martin Wehrle, and Malte Helmert. An analysis of merge strategies for merge-and-shrink heuristics. In *Proc. ICAPS 2016*, pages 294–298, 2016. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13148.

39 Álvaro Torralba and Silvan Sievers. Merge-and-shrink task reformulation for classical planning. In *Proc. IJCAI 2019*, pages 5644–5652, 2019. doi:10.24963/IJCAI.2019/783.

40 Son Cao Tran, Enrico Pontelli, Marcello Balduccini, and Torsten Schaub. Answer set planning: A survey. *Theory and Practice of Logic Programming*, 23(1):226–298, 2023. doi:10.1017/S1471068422000072.

41 Peter van Beek and Xinguang Chen. CPlan: A constraint programming approach to planning. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI 1999)*, pages 585–590. AAAI Press, 1999. URL: http://www.aaai.org/Library/AAAI/1999/aaai99-083.php.

42 Damien Van Meerbeeck, Arnaud Lequen, Gilles Pesant, and Jendrik Seipp. Code for the CP 2026 paper "An Automata-Based Constraint Programming Framework for Optimal Classical Planning". https://doi.org/10.5281/zenodo.20081550, 2026.

43 Alessandro Zanarini, Gilles Pesant, and Michela Milano. Planning with soft regular constraints. In *ICAPS 2006 Workshop on Preferences and Soft Constraints in Planning*, pages 73–78, 2006.