An Automata-Based Constraint Programming Framework for Optimal Classical Planning (Extended Abstract)

Damien Van Meerbeeck[1], **Arnaud Lequen**[1], **Gilles Pesant**[2], **Jendrik Seipp**[1]

[1]Linköping University, Sweden
[2]Polytechnique Montréal, Canada
damien.van.meerbeeck@liu.se, arnaud.lequen@liu.se, gilles.pesant@polymtl.ca, jendrik.seipp@liu.se

Introduction

Optimal classical planning is the challenge of finding the shortest sequence of actions (called a *plan*) that transforms a given initial situation into one that satisfies a given goal description in a deterministic environment (Ghallab, Nau, and Traverso 2004). Most of today's strongest optimal classical planners are based on state-space search with admissible heuristics (Helmert and Domshlak 2009).

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 (Rossi, Van Beek, and Walsh 2006), which makes it easy to enrich a planning encoding with additional constraints that strengthen propagation without changing the set of valid plans.

Our approach builds on recent automata-based representations of planning tasks. Starting from an input planning task, we automatically derive a factored set of deterministic finite automata and use them as the backbone of a CP model. Finding a plan is then reduced to finding a word accepted by all automata through REGULAR (Pesant 2004) constraints. Unlike previous automata-based CP approaches, which rely on manually engineered automata, our framework derives the atomic automata automatically from the planning model itself. This turns automata-based CP planning into a general pipeline rather than a domain-specific modeling task.

The dominance of heuristic search in optimal classical planning is largely due to the wide array of admissible heuristics available. Since CP provides a rich modeling language, we show how to encode information underlying such heuristics from the planning literature directly into our framework, in the form of redundant constraints that strengthen propagation. Concretely, we focus on two families of constraints from the operator-counting framework (Pommerening et al. 2014): landmark constraints and net change constraints.

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

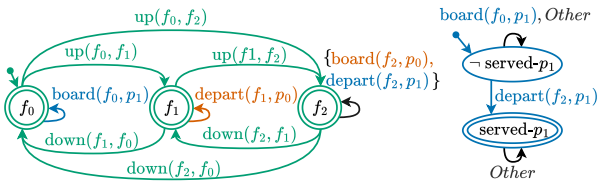

Figure 1: Example automata of two state variables of a planning task describing the problem of transporting passengers of an elevator to their destination floors.

Background

Classical Planning We follow standard definitions and only recall notations here. 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 $v \in V(s)$.

Each action $a \in A$ is a pair $a = \langle pre(a), eff(a) \rangle$, where $pre(a)$ and $eff(a)$ are partial states called the *precondition* and *effect* of a. Applying a in s yields the state $s[\![a]\!]$. State s_I is the *initial state* and the partial state G is the *goal*.

Deterministic Finite Automata We use the standard definitions for automata (Hopcroft, Motwani, and Ullman 2007). 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*.

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 $\delta(s, a) = s[\![a]\!]$ iff a is applicable in state s.

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 *atomic representation* of Π is the factored representation $S = \{\mathcal{A}^{v_1}, \ldots, \mathcal{A}^{v_{|V|}}\}$, where each automaton $\mathcal{A}^{v_i}$ is induced by the projection of the planning task on v_i. Figure 1 shows two such automata for an example task where an elevator has to transport people to their destination floor.

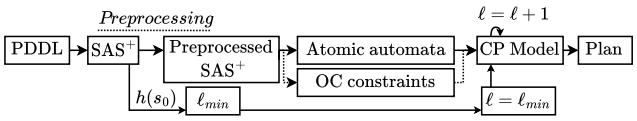

Figure 2: Pipeline of our approach. Optional components of the pipeline are associated with dotted lines, and are enabled or disabled in our experiments to evaluate their impact.

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, $\mathcal{C}$ is a finite set of constraints over the variables, and each variable $x \in \mathcal{X}$ has a finite domain $\mathcal{D}(x) \subseteq \mathcal{D}$.

We 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 $s = x_1 + \cdots + x_n$.

Planning as CP with Automata

Our approach reformulates a planning task as a collection of automata and then compiles this representation into a CP model. We describe our contribution in this section, inspired by the CP model of Babaki et al. (2020).

Algorithm Outline Figure 2 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.

The SAS$^+$ task is then preprocessed using an off-the-shelf tool (Alcázar and Torralba 2015). Next, we convert the preprocessed task into its atomic factored representation, that is, the set of automata $S = \{\mathcal{A}^{v_1}, \ldots, \mathcal{A}^{v_{|\mathcal{V}|}}\}$ introduced in the previous section. From the same representation, we also derive the operator-counting constraints discussed later.

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 of horizon smaller than $\ell_{\min}$ is infeasible and horizons are considered in increasing order, the first solution found is optimal in plan length.

Preprocessing The preprocessing phase modifies the SAS$^+$ task before the automata are generated. It uses an off-the-shelf preprocessor (Alcázar and Torralba 2015) to prune irrelevant atoms and actions, and enrich the task with implied atoms.

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. For additional details about the computation of these implied atoms, we refer to Alcázar et al. (2013).

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 components can have more ambiguous effects on CP performance, so we evaluate them separately and jointly.

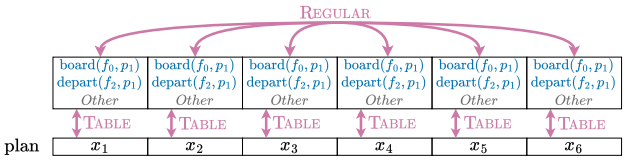

Figure 3: Constraints added for the rightmost automaton of Figure 1 for a plan of length $\ell = 6$. This illustrates the reduction to the reduced alphabet, and the posting of the constraints.

Base CP Model Our base CP model encodes the bounded planning problem for a fixed horizon ℓ. Following Babaki et al. (2020), it uses REGULAR constraints to enforce the dynamics of the automata in the atomic representation. For each $k \in \{1, \ldots, n\}$, 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 $\mathit{plan} = \langle x_1, \ldots, x_\ell \rangle$, such that $\mathcal{D}(x_i) \subseteq A$;
- For each automaton $\mathcal{A}^k$, a sequence $\mathit{plan}^k = \langle x_1^k, \ldots, x_\ell^k \rangle$ 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:

$$\text{REGULAR}(\langle x_1^k, \ldots, x_\ell^k \rangle, \mathcal{A}^k) \quad \forall k,\ 1 \leq k \leq n$$

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,

$$\text{TABLE}(\langle x_i, x_i^k \rangle, \mathcal{T}^k) \quad \forall k,\ 1 \leq k \leq n,\ \forall i,\ 1 \leq i \leq \ell,$$

where $\mathcal{T}^k = \{\langle a, \sigma^k(a) \rangle \mid a \in A\}$. Figure 3 summarizes, 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.

Counting Actions

While factored representations enable efficient propagation via REGULAR constraints, they localize information that could provide additional pruning when combined across variables. We address this by incorporating *operator-counting constraints* (Pommerening et al. 2014), which are linear inequalities over action occurrence counts α_a that strengthen our BASE model with global information. We present landmark and net change constraints, then show how to compile them compactly by grouping action counts.

Landmark Constraints A *disjunctive action landmark* (Helmert and Domshlak 2009) is a set $L \subseteq A$ such that at least one action in L must appear in every plan. We use an off-the-shelf tool to compute landmarks (Helmert and Domshlak 2009), and each landmark $L \subseteq A$ is included in our model through the following *landmark constraint*:

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

Net Change Constraints *Net change constraints* (Pommerening et al. 2014) track the number of changes of individual atoms. The net change of an atom $\langle v, d \rangle$ is $\delta_{\langle v,d \rangle} \in \{-1, 0, 1\}$, and represents the difference between the number of times the atom must be established and the number of times it must be strictly deleted.

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 consider the sets of actions $PA_{\langle v,d \rangle}$ that respectively *possibly add* atom $\langle v, d \rangle$, and $SD_{\langle v,d \rangle}$ that *surely delete* atom $\langle v, d \rangle$. We then obtain constraints of the following form:

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

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 factorizes 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. Whenever a set contains actions that occur with different coefficients in the current constraint, we split that set accordingly. For each constraint, actions in the current partition are regrouped based on their coefficients (-1, 0, or 1), splitting any set that contains actions with different coefficients. 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.

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

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

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

$$\text{SUM}(\ell, \{\vartheta_1, \vartheta_2, \ldots, \vartheta_m\})$$

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

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 $\{1, \ldots, \ell\}$, and the constraint is posted as

$$\text{SUM}(\vartheta^L, \bigsqcup\nolimits_{\substack{1 \leq i \leq m \text{ s.t.} \\ D_i \subseteq L}} \{\vartheta_i\}). \tag{1}$$

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 $\{0, \ldots, \ell\}$ representing the corresponding sums, defined as follows for $\lambda \in \{\lambda_p, \lambda_n\}$:

$$\text{SUM}(\vartheta^\lambda, \bigsqcup\nolimits_{\substack{1 \leq i \leq m \text{ s.t.} \\ D_i \subseteq \lambda}} \{\vartheta_i\})$$

We then post the grouped net change constraint. For an atom $\langle v, d \rangle$, we post the inequality $\vartheta^{\lambda_p} - \vartheta^{\lambda_n} \geq \delta_{\langle v,d \rangle}$.

Experiment Setup

Starting from PDDL tasks, we use Scorpion (Seipp, Keller, and Helmert 2020) to generate SAS$^+$ representations and compute the lower bound $\ell_{\min}$ using the LM-Cut heuristic (Helmert and Domshlak 2009). Preprocessing optionally adds implied atoms, followed by the computation of atomic representations and operator-counting constraints.

We build our CP models using CP-SAT (Perron and Didier 2025), then iteratively solve for increasing plan lengths $\ell \geq \ell_{\min}$ until finding a solution.

We evaluate our approach on all IPC optimal track benchmarks (47 domains adapted to have unit costs) using Intel Xeon Gold 6130 processors with a 30-minute cutoff and a memory limit of 8 GiB.

We compare against three baselines: CP-based GP-WCSP (Cooper, de Roquemaurel, and Régnier 2011) adapted to use CP-SAT, (2) manual automata from Babaki et al. (2020) integrated into our model, and (3) heuristic-search baselines from IPC: blind search (BLIND) and A* with LM-Cut (LM-CUT) (Helmert and Domshlak 2009).

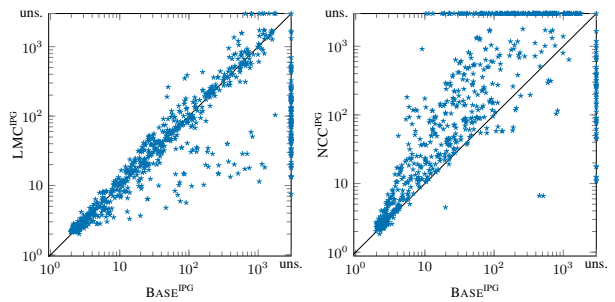

Figure 4: Comparison of overall runtime for BASE^{IPG} against LMC^{IPG} and NCC^{IPG}. BASE^{IPG} is our base model, LMC^{IPG} is our model with landmark constraints, and NCC^{IPG} is with net change constraints. All models are enriched with implied preconditions and goals. Points on the "uns." axes indicate runs that hit the time or memory limit.

	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**

Table 1: 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.

Experiment Results

Impact of Task Preprocessing We evaluate preprocessing effects on the base model BASE using three variants: BASE^{IP} (implied preconditions), BASE^{IG} (implied goals), and BASE^{IPG} (both). The combined variant BASE^{IPG} performs best, solving 690 tasks and outperforming the base model in 7 domains while never losing. The two preprocessing techniques are complementary, as their combined effect strictly dominates each individual technique.

Impact of Redundant Constraints We evaluate the effect of adding two families of redundant constraints to the strongest preprocessed model BASE^{IPG}: landmark constraints LMC^{IPG} and net change constraints NCC^{IPG}.

Landmark constraints are the most successful extension (Table 1). While these constraints increase runtime for some instances (Figure 4), they often provide sufficient pruning to compensate for this overhead.

Net change constraints perform substantially worse overall, as they induce finer action set partitions that increase occurrence variables and memory usage (876 tasks hit memory limits vs. 454 for the base model). However, they outperform other approaches in four domains characterized by highly interdependent actions that must be applied in specific orders.

Comparison with Baselines Table 1 shows LMC^{IPG} clearly outperforms the CP-based baseline GP-WCSP, supporting the effectiveness of automata-based CP. Note however that GP-WCSP was designed for arbitrary action costs while we focus on unit-cost planning.

As a second CP-based reference point, we compare against the manually designed automata of Babaki et al. (2020). These automata were proposed for three domains: *Floortile*, *Miconic* and *Scanalyzer*. When we replace our automatically generated automata with theirs, the number of solved tasks decreases overall, although both approaches perform similarly in *Floortile*. Babaki et al. solve 51 instances in *Miconic*, 21 in *Scanalyzer*, and 6 in *Floortile*. By comparison, LMC^{IPG} solves 102 *Miconic* instances, 25 *Scanalyzer* instances, and 5 *Floortile* instances. 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 run them here on CP-SAT with its default search.

Our approach does not yet match the overall performance of the heuristic-search baselines BLIND and LM-CUT. Nevertheless, it remains competitive in a meaningful subset of domains, which points to complementary strengths rather than a simple domination relation.

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 favorable combination of factors. First, the lower bound $\ell_{\min}$ that we compute is often very close to the optimal plan length ℓ^*, which means that our algorithm rarely has to prove infeasibility for many 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.

Conclusions

We presented an end-to-end, automated CP framework for optimal classical planning, which leverages an automata-based representation of the planning task.

We showed that our 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

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting Regression in Planning. In (Rossi 2013), 2254–2260.

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Babaki, B.; Pesant, G.; and Quimper, C. 2020. Solving Classical AI Planning Problems Using Planning-Independent CP Modeling and Search. In Harabor, D.; and Vallati, M., eds., *Proceedings of the 13th Annual Symposium on Combinatorial Search (SoCS 2020)*, 2–10. AAAI Press.

Cooper, M. C.; de Roquemaurel, M.; and Régnier, P. 2011. A weighted CSP approach to cost-optimal planning. *AI Communications*, 24(1): 1–29.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2007. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley. ISBN 978-0-321-47617-3.

Perron, L.; and Didier, F. 2025. CP-SAT. https://developers.google.com/optimization/cp/cp_solver/. Version v9.12, Google.

Pesant, G. 2004. A Regular Language Membership Constraint for Finite Sequences of Variables. In Wallace, M., ed., *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *Lecture Notes in Computer Science*, 482–495. Springer-Verlag.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based Heuristics for Cost-optimal Planning. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 226–234. AAAI Press.

Rossi, F., ed. 2013. *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. AAAI Press.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *Journal of Artificial Intelligence Research*, 67: 129–167.