Solving Classical AI Planning Problems Using Planning-Independent CP Modeling and Search

Behrouz Babaki and **Gilles Pesant**
Polytechnique Montréal, Canada
{behrouz.babaki, gilles.pesant}@polymtl.ca

Claude-Guy Quimper
Université Laval, Canada
claude-guy.quimper@ift.ulaval.ca

Abstract

The combinatorial problems that constraint programming typically solves belong to the class of $\mathcal{NP}$-hard problems. The AI planning commun ity focuses on even harder problems: for example, classical planning is $\mathcal{PSPACE}$-hard. A natural and well-known constraint programming approach to classical planning solves a succession of fixed plan-length problems, though to date it has had limited success. We revisit this approach in light of recent progress on general-purpose branching heuristics. We conduct an empirical comparison to show the importance of using effective combinatorial search heuristics with this approach and that the quality of the plans produced is sometimes comparable to that of state-of-the-art planners.

Introduction

While there is a history of more than twenty years in the application of SAT, CP, and MIP to AI planning problems, there has been considerable renewed interest in a deeper cross-fertilization between these areas, as evidenced by a number of recent papers presented at the ICAPS, AAAI, and IJCAI conferences, by a 2018 Dagstuhl seminar (Beck et al. 2018) on that topic, by the co-location of the 2018 edition of the ICAPS and CPAIOR international conferences, and by an incubator workshop on Constraints and AI Planning as part of the 2019 edition of the CP conference.

The combinatorial problems that Constraint Programming (CP) typically solves belong to the class of $\mathcal{NP}$-hard problems. The AI planning community focuses on even harder problems. Consider the Classical Planning Problem, perhaps the simplest and most thoroughly investigated in all of AI planning, and which is $\mathcal{PSPACE}$-hard: given a unique initial state of the world, apply a sequence of actions in order to reach a goal state. Figure 1 shows the classic example of Blocks World: starting from the initial state, a sequence of `stack`, `unstack`, `pickup`, `putdown` actions are performed on blocks `a`, `b`, and `c` in order to reach the desired goal state. Note that we do not know in advance how many actions need to be applied. A natural satisfiability approach to classical planning, known for over twenty years, is to solve a succession of fixed plan-length instances.

Figure 1: An instance of Blocks World: initial state (left) and goal state (right).

This basic problem can be extended and complexified in many ways — to name a few: actions may have a duration and need not be deterministic, leading to several possible outcomes; there may be several agents taking actions concurrently; the state of the world may only be partially observable and the state space may not be finite or even discrete.

Within the AI planning academic community the focus is largely on the design of *domain-independent* planners, meaning that they should be able to solve problems from a wide range of domains: expressed in a modeling language such as STRIPS, SAS+, or PDDL, they are given as input a set of objects and predicates with which a state of the world can be described and a set of actions together with their preconditions, effects, and costs — this is called the *planning domain* — as well as a specification of the actual instance in the form of initial and goal states, and possibly a subset of allowed actions and an objective to optimize — this is called the *planning problem*.

In contrast to CP, where modeling is typically a non-trivial task that can greatly impact how efficiently we solve the problem at hand, here the modeling task should be straightforward without injecting any *domain knowledge* from the human modeler. From that perspective the care being put into writing a good CP model would probably be considered by many from that community as not being domain-independent. We aim to show that designing a competent CP model for classical planning can be fairly straightforward as well , irrespective of the planning domain. In a change of focus, we also claim to be considering *planning-independent* solvers since we will use generally-available CP constructs without anything specific to planning problems.

Looking at the scientific literature on CP-based planners, the consensus seems to be that while some progress has

been made, they generally still haven't reached the performance of state-of-the-art planners, except in specialized domains. Here we revisit classical CP modeling for planning with recent improvements to search heuristics. We show that a simple planning-agnostic model can perform rather well compared to state-of-the-art planners that include much planning-specific sophistication.

The main contributions of this paper are: to give a step-by-step process to build a CP model for classical planning that could be automated given a planning domain; to identify some general-purpose search heuristics developed for CP but that perform much better than others for planning; to show that a CP approach even without planning knowledge may yield competitive plans.

In the next section we give a brief introduction to Constraint Programming focusing on the most relevant constraints for classical planning and on certain search heuristics originating from CP which, as we will soon discover, work particularly well for planning. We then review previous work related to CP and planning. Next we introduce a natural CP-based approach to planning. We then apply our approach to three well-known planning domains and present an empirical evaluation of it.

Constraint Programming

In CP, a combinatorial problem is represented as a *Constraint Satisfaction Problem* using a finite set of discrete variables $X = \{x_1, x_2, \ldots, x_n\}$ each taking its value from a finite domain, $x_i \in D_i \subset \mathbb{Z},\ 1 \leq i \leq n$, and a finite set of constraints $C = \{c_1, c_2, \ldots, c_m\}$ each expressed on a subset of the variables $c_j(x_{j_1}, x_{j_2}, \ldots, x_{j_k}) \subset \mathbb{Z}^k,\ 1 \leq j \leq m$. One must find a combination of values from the domain of each variable that simultaneously satisfies every constraint. When faced with a combinatorial *optimization* problem, a cost variable to optimize over is added to the model together with a constraint linking its value to the cost of each assignment to X.

Much of the computational strength of CP comes from the use of powerful filtering algorithms associated with each type of constraint that efficiently remove infeasible variable-value assignments, thus greatly shrinking the combinatorial search space to explore. It does so by exploiting the combinatorial structure of each type of constraint instead of considering them merely as general relations. One such type of constraint that will be featured prominently here enforces regular language membership for a sequence of variables.

Automata-Based Constraints

Recall that an automaton A is defined by a tuple $\langle \Sigma, \mathcal{Q}, q_0, F, \delta \rangle$ where Σ is an alphabet, $\mathcal{Q}$ a set of states, $q_0 \in \mathcal{Q}$ an initial state, $\mathcal{F} \subseteq \mathcal{Q}$ a set of final states, and $\delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ a set of transitions. A word $w \in \Sigma^n$ is recognized by automaton A if there exist states $q_1, \ldots, q_n$ such that $\langle q_{i-1}, w_i, q_i \rangle \in \delta$ for all $i \in \{1, \ldots, n\}$ and $q_n \in \mathcal{F}$. Constraint REGULAR(X, A) holds if the values taken by the sequence of variables X spell out a word belonging to the regular language described by automaton A (Pesant 2004). Constraint COSTREGULAR (Demassey, Pesant, and Rousseau 2006) is a generalization of REGULAR that associates a cost with each transition of the automaton. The constraint is supplied with a matrix giving such costs and with a cost variable that corresponds to the sum of the costs of the transitions used by the word.

Internally each of these constraints maintains a layered digraph, built by unfolding the automaton, which is a compact representation of its solution set that can be used to filter out inconsistent assignments efficiently. Each solution to the constraint (i.e. each word assembled from the domains of the variables and recognized by the automaton) is in one-to-one correspondence with a path between a pair of vertices from the initial layer to the last layer in that graph. One can easily compute and maintain such feasible paths (and remove any arc that is not traversed by a path), the number of partial paths to and from any given vertex, and the length of shortest and longest weighted (partial) paths.

The identification of feasible paths enables us to filter out infeasible variable-value assignments (i.e. those that do not appear in any such path). In the case of COSTREGULAR if we bound the cost variable from above or even from below, the length of shortest and longest weighted paths is used to restrict further the set of words allowed by the constraint and filter out even more variable-value assignments.

Counting-Based Search Heuristics

Besides reducing the combinatorial search space through domain filtering algorithms associated to each constraint, the other important aspect in CP is guiding the exploration of the search space using heuristics that define how we branch while building the search tree. Several domain-independent search heuristics have been proposed though search is also programmable in CP. *Counting-based search* (Pesant, Quimper, and Zanarini 2012) is a domain-independent heuristic that counts how often each variable-value assignment appears in a solution local to a given constraint and combines that information from each constraint in order to branch on a variable/value assignment that is most likely to be featured in a global solution. For the automata-based constraints this count is achieved by multiplying the number of partial paths at each end of an arc representing a variable-value assignment (Zanarini and Pesant 2009).

Related Work

We first review problem representation. Early work on applying CP to AI planning was mostly concerned with parallel planning and only used elementary logical constraints over Boolean variables (e.g. (Do and Kambhampati 2001)(Lopez and Bacchus 2003)). One exception is (van Beek and Chen 1999) who used (non-binary) finite-domain variables but still simple constraints. For temporal planning (Vidal and Geffner 2006) combine the strength of Partial Order Causal Link planners' branching scheme with the pruning abilities of CP. For every potential action in a plan, they define starting-time and presence variables reminiscent of *interval variables* introduced later in CPO[1] for constraint-based

[1] www.ibm.com/analytics/cplex-cp-optimizer

scheduling. Again here the constraints are simple: disjunctions, logical implications, and simple temporal constraints. (Zanarini, Pesant, and Milano 2006) propose a higher-level CP model taking advantage of the stronger inference offered by global constraints, describing the effects of actions on the state of a planning object as an automaton and then enforcing a valid plan by stating a REGULAR constraint for each object. They also consider the soft variant of these constraints in order to express preferences between goal states and to improve search guidance. (Barták and Toropila 2008) reformulate some of the early CP models in a more compact way by using TABLE (Bessière and Régin 1997) constraints to encode state transitions of objects under planning actions. This can be seen as a decomposition of the REGULAR constraints featured in the model of the previous paper. In their constraint-based planner (Gregory, Long, and Fox 2010) exploit the structure of the planning problem to identify recurring patterns of actions, called macro-actions, that can be reused instead of being reconstructed each time. (Judge and Long 2011) define meta-variables that correspond to subsets of the original variables each related to part of the goal state description and whose domain contains the actions that achieve that part of the goal. More recently (Ghooshchi et al. 2017) use TABLE constraints but allow wildcards in tuples for a more concise representation of the transitions.

In terms of search guidance, very often general-purpose CP heuristics have been used. (van Beek and Chen 1999) apply the *min-dom* dynamic variable ordering and conflict-directed backjumping, (Barták and Toropila 2008) use static variable and value orderings: *reverse lexicographic* for variables and *lexicographic* for values. As an exception, (Judge and Long 2011) design a goal-directed variable and value ordering heuristic that exploits the structure of the planning problem. But even recent planners such as (Ghooshchi et al. 2017) use *min-dom* and *dom/wdeg*.

A Natural CP Approach to Classical Planning

Given a fixed plan length ℓ (thus turning a $\mathcal{PSPACE}$-hard problem into a $\mathcal{NP}$-hard one), we can model a classical planning problem in CP by defining:

i.. a sequence of variables $\langle x_1, x_2, \ldots, x_\ell \rangle$, one for every step of the plan;

ii.. a domain of actions D, with $x_i \in D,\ 1 \leq i \leq \ell$;

iii.. a REGULAR (or COSTREGULAR) constraint for each object, from the automaton describing the effects of actions on the state of that object (i.e. factored transition systems).

Going back to our example at Figure 1, we can fix ℓ to 6, define the domain of actions $D = \{\mathit{stack}(B, B'), \mathit{unstack}(B, B'), \mathit{pickup}(B), \mathit{putdown}(B) \mid B, B' \in \{a, b, c\}, B \neq B'\}$, and add constraints $\text{REGULAR}(\langle x_1, x_2, \ldots, x_\ell \rangle, \mathcal{A}_\mathtt{a})$, $\text{REGULAR}(\langle x_1, x_2, \ldots, x_\ell \rangle, \mathcal{A}_\mathtt{b})$, and $\text{REGULAR}(\langle x_1, x_2, \ldots, x_\ell \rangle, \mathcal{A}_\mathtt{c})$, one for each block with its respective automaton as illustrated at Figure 2. Note that the automata have an identical structure but different initial and accepting states, reflecting the initial and goal states of the planning problem. Any instantiation that simultaneously satisfies all the constraints, such as

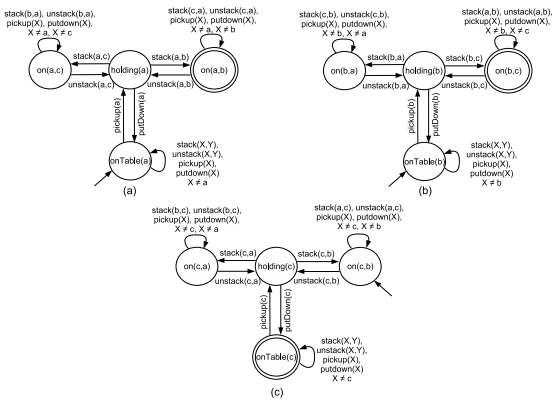

Figure 2: Automata for blocks a, b, and c of the instance at Fig. 1 (reproduced from (Zanarini, Pesant, and Milano 2006)).

$\langle \mathit{unstack}(c, b), \mathit{putdown}(c), \mathit{pickup}(b), \mathit{stack}(b, c), \mathit{pickup}(a), \mathit{stack}(a, b) \rangle$, corresponds to a valid plan.

Algorithm 1: Solving models of increasing length

```
1 c* ← ∞
2 for ℓ ← ℓ_min to ∞ do
3 |   if c_min(ℓ) ≥ c* then
4 |   |   return c*
5 |   c ← solve(model(ℓ), T_max)
6 |   if c < c* then
7 |   |   c* ← c
```

Algorithm 1 repeatedly solves the fixed-length CP model for increasing plan length ℓ, starting at some possibly trivial lower bound (line 2). Let $c_{\min}()$ be some nondecreasing function of plan length ℓ that provides a lower bound on the cost of plans of length ℓ. At each length we check the current best cost $c^\star$ against that lower bound and end the search if a cheaper plan cannot be found at longer plan lengths (lines 3-4). We attempt to solve each fixed-length model up to a given time limit $T_{\max}$ (as was done e.g. in (Barták and Toropila 2008)) to avoid possibly spending all our time proving that no plan exists at some intermediate length (line 5). If we time out at some plan length, we return the cost of the best plan found, or ∞ if none was found. If this improves the current best cost, we update it (lines 6-7).

This natural way of formulating classical planning problems in CP benefits from the usual flexibility and extendability of its models as well as from the filtering of the variables' domains through the constraints, ruling out some actions at various steps of the plan whereas AI planners may tend to concentrate on the next step. It also offers powerful general-purpose search heuristics that exploit the combinatorial structure of these large-arity constraints and whose performance we evaluate later on. However an important drawback of this approach is its reliance on a fixed plan length. If we solve a succession of instances of increasing plan length, finding an optimal plan requires proving the in-

feasibility of each intermediate plan length, which may be very time consuming, and if we start from a lower bound on feasible plan length $\ell_{\min}$ that is far from the shortest feasible one we will spend most of the time searching over infeasible plan lengths even if we set a time limit $T_{\max}$.

An Elevator Problem

In planning domain `miconic` (Koehler and Schuster 2000), a number of passengers should be transported between different floors of a building. The elevator can go up or down between floors and at each floor passengers can enter or leave the elevator. For each passenger we are given an origin and a destination floor. We are also given the initial floor f_0 the elevator is on.

There are two kinds of actions:

i.. for each passenger p, a *board*(p) and *depart*(p) action;

ii.. for each floor f, an *up*(f) and *down*(f) action, f representing the effect of the action (i.e. new current floor).

Given n_p passengers and n_f floors, we have $2n_p + 2(n_f - 1)$ actions in all. (Note that there is no *up* action from the top floor nor *down* action from the bottom one.) Despite the original PDDL description including the current floor as an additional parameter of each of these actions, we do not need it here because it will already be reflected in the states of our automata. This contrasts our representation with state transition graphs which are automatically generated from the PDDL description. For example, the number of actions in a SAS+ encoding of the `miconic` domain is more than one order of magnitude larger than in our representation.

The natural planning objects whose state is affected by actions are the elevator and each passenger. Our elevator automaton $\mathcal{A}_1$ has n_f states representing the current floor. All states are accepting and the initial state is f_0. At each given state/floor, the *up/down* actions perform the obvious transitions if they are consistent with the relative position of the current and new floors (e.g. one cannot go up to a lower floor) and are forbidden otherwise. The *board/depart* actions are irrelevant to the elevator and so they loop at each state. The resulting automaton is shown at Figure 3.

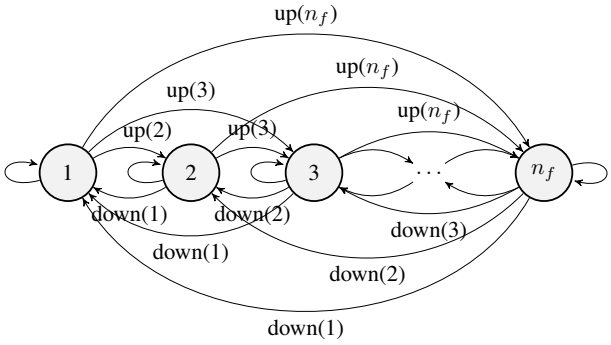

Figure 3: Elevator automaton. Loops are on *board/depart* actions.

For each passenger p with origin/destination pair $\langle o_p, d_p \rangle$, we can create an automaton with five states (`waiting, can board, boarded, can depart, served`), the first being the initial state (unless $o_p = f_0$ in which case it starts at state `can board`, or $o_p = d_p$ in which case it trivially starts at `served`) and the last, the accepting one. The structure of this automaton is the same for all passengers but the effect of particular actions differs: e.g. for p an *up*(o_p) action while in state `waiting` will transition to state `can board` whereas *up*(f) to any other floor f will loop. Therefore we define a single passenger automaton $\mathcal{A}_2$ on new actions $D^{pass.} = \{$*board, depart, someoneElse, atOrig, atDest, somewhereElse*$\}$, but define for each p an appropriate map $M_p : D \mapsto D^{pass.}$ from the original actions to these new ones: *board*(p) to *board*, *board*(p') to *someoneElse*, ..., *up*(o_p) to *atOrig*, and so forth. The resulting automaton is shown at Figure 4.

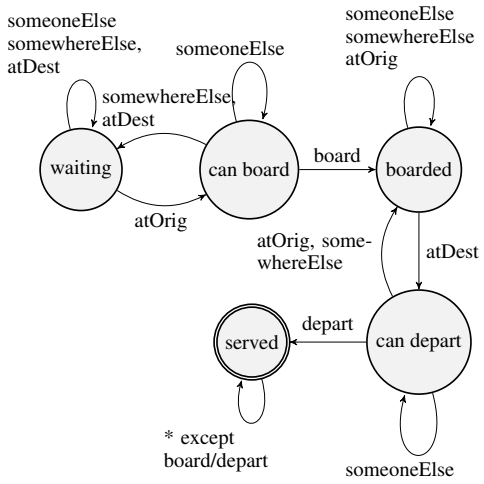

Figure 4: Passenger automaton

Let $\mathcal{P}$ be the set of passengers. Constraints (1) to (5) define our simple model. Constraint (1) enforces the elevator automaton on the main action variables. Constraints (2) link the main variables to the auxiliary action variables for each passenger on which the passenger automaton is enforced through Constraints (3).[2]

$$\text{REGULAR}([x_1, \ldots, x_\ell], \mathcal{A}_1) \quad (1)$$

$$x_i^p = M_p(x_i) \qquad \forall\, 1 \le i \le \ell,\ p \in \mathcal{P} \quad (2)$$

$$\text{REGULAR}([x_1^p, \ldots, x_\ell^p], \mathcal{A}_2) \qquad \forall\, p \in \mathcal{P} \quad (3)$$

$$x_i \in D \qquad \forall\, 1 \le i \le \ell \quad (4)$$

$$x_i^p \in D^{pass.} \qquad \forall\, 1 \le i \le \ell,\ p \in \mathcal{P} \quad (5)$$

Because all actions here have unit cost, plan cost equals plan length so $c_{\min}(\ell) = \ell$. A simple lower bound $\ell_{\min}$ on plan length is $2n_p$ (each passenger must board and depart) plus the number of distinct values among $(\{o_p \mid p \in \mathcal{P}\} \setminus \{f_0\}) \cup \{d_p \mid p \in \mathcal{P}\}$ (the set of floors that must be

[2] Constraints (2)-(3) are actually combined for each p so that the information computed by (3) for counting-based search is transferred to the main variables (Pesant and Zanarini 2011).

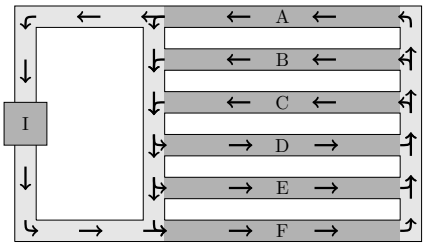

Figure 5: An instance of the greenhouse logistics problem (reproduced from (Helmert and Lasinger 2010)).

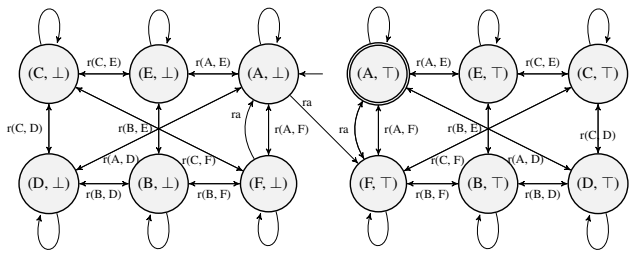

Figure 6: Automaton for instance `p03`. Loops accept any action that is not associated to an outgoing transition. States are labeled with tuples (p, a) where p is the position and $b = \bot$ means the batch was not analyzed and $b = \top$ means the batch was analyzed. The initial and accepting states, identified respectively by a short arrow and a double circle, are for the batch on segment A.

visited). This bound is rather trivial and could be improved. However, we decided to introduce as little domain knowledge as possible into our models.

We could add to our model some symmetry-breaking constraints about the order in which passengers board and depart by imposing that in between *up/down* actions in a plan, all *depart* actions appear in increasing order of p and then all *board* actions in increasing order. But because such symmetry breaking appears to interfere with our search heuristics,[3] we do not include it in our model.

A Greenhouse Logistics Problem

We consider next an automated greenhouse logistics problem. The `scanalyzer` planning domain (Helmert and Lasinger 2010) models the problem of automated greenhouse logistic management. Smart greenhouses are equipped with imaging facilities that collect data about the plants being grown. Plants are transported between the greenhouses and the imaging facilities by a system of conveyor belts. Plants are grouped in batches, each located on a distinct segment (A to F in Figure 5). Each batch must go through an imaging facility (I) at least once and return to its original position. A given set of *rotate* (*r*) actions exchange two batches on segments and another set of *rotate&analyze* (*ra*) actions does this while sending one of the batches through the imaging facility. The latter actions are considered more expensive to perform than the former. The objective is to find a plan of minimum cost.

The instance at Figure 5 allows actions that swap the batch on any segment among A, B, and C with the one on any segment among D, E, and F (nine actions) and one that swaps the batches on segments A and F while sending the batch from A through the imaging facility. An optimal solution consists of six *rotate&analyze* actions and eight *rotate* actions, e.g. $\langle$ *ra*(A,F), *r*(A,E), *ra*(A,F), *r*(A,D), *ra*(A,F), *r*(A,E), *ra*(A,F), *r*(B,F), *r*(C,D), *r*(A,D), *ra*(A,F), *r*(C,F), *ra*(A,F), *r*(B,F)$\rangle$.

Note that in the PDDL description of this problem, the actions are parameterized by batches currently located on the rotating segments. As a result, the number of actions in SAS+ encodings of problems in this domain is up to four orders of magnitude higher than in our encoding.

[3]This is known to happen in combinatorial search.

Planning objects are the batches of plants. It is possible to use an automaton to keep track of the state of the batches. Automaton $\mathcal{A}_3$ depicted at Figure 6 shows how the batch initially located on segment A changes states upon an action. Each state is labeled with a tuple (location, analyzed) where *location* is the segment on which the batch lies and *analyzed* is a Boolean indicated whether the batch was analyzed. The initial state is $(A, \bot)$ since the batch is initially on segment A and is not analyzed and the final state is $(A, \top)$, since the batch must return to its origin while being analyzed. The automata for the other batches are the same except for the initial and final states.

Let $\mathcal{B}$ be the set of batches. Each action $a \in D$ has a cost of $C_a \geq 1$. We have $C_a = 1$ if it is a *rotate* action and $C_a = 3$ if it is a *rotate and analyze* action. Constraints (6) to (9) define the model.

$$\text{minimize } z \tag{6}$$
$$z \geq 2|\mathcal{B}| + \ell \tag{7}$$
$$\textsc{CostRegular}([x_1, \ldots, x_\ell], \mathcal{A}_3, C, z) \quad \forall\, b \in \mathcal{B} \tag{8}$$
$$x_i \in D \quad \forall\, 1 \leq i \leq \ell \tag{9}$$

The cost variable z is made equal to $\sum_{i=1}^{\ell} C_{x_i}$ through constraints (8). Since each batch must be analyzed at least once at a cost of 3 and each other action in the plan has a cost of 1, constraint (7) provides a trivial lower bound on the objective function of $3|\mathcal{B}| + 1 \cdot (\ell - |\mathcal{B}|)$. Constraints (8) enforce the batch automaton over the action variables for each batch.

We apply Algorithm 1 using lower bound on plan length $\ell_{\min} = |\mathcal{B}|$, the number of batches, since each of them needs to be analyzed and we cannot analyze several batches at a time, and lower bound function on plan cost $c_{\min}(\ell) = 2|\mathcal{B}| + \ell$, as previously established. As for the previous domain, these trivial bounds aim at introducing as little domain knowledge as possible.

A Floor Tile Painting Problem

In the planning domain `floortile`[4], a set of robots paint tiles according to a certain colour pattern. The robots can move to an adjacent tile in four directions (up, down, right, and left) unless that tile is painted or occupied. They can also colour the tile above or below them if it is not painted or occupied using their spray gun, or change the spray to another colour. We are given the initial location and spray gun colour of each robot, the relative locations of the tiles, and which tile should be painted in a given colour. There are three kinds of actions:

i.. for each robot r and colour c, a *change-colour*(r, c) action;

ii.. for each robot and to-be-painted tile t, a *paint-up*(r, t) and *paint-down*(r, t) action;

iii.. for each robot and current tile, *move-up*(r, t), *move-down*(r, t), *move-left*(r, t), and *move-right*(r, t) actions.

Given n_r robots, n_c colours, and n_t tiles, we have $n_r \cdot (n_c + 6n_t)$ actions in all. Again we simplify the parameter list of the actions from the original PDDL description when some of them are already reflected in the states of our automata. This in turn reduces the number of actions in our representation compared to the state transition graphs which are automatically generated from the PDDL description.

The natural planning objects whose state is affected by actions are the robots and the tiles. The robot automaton $\mathcal{A}_4$ is structured similarly to the one for greenhouse logistics: $n_t \times n_c$ states, one per location/colour pair, and the obvious transitions between locations and colours. For paint actions, an allowed one (i.e. currently at the right location and with the right colour) loops and all others are forbidden. Any action for the other robots loops. Therefore its set of actions can be reduced to $D^{robot} = \{$ *change-colour(c), paint-up(t), paint-down(t), move-up(t), move-down(t), move-left(t), move-right(t), otherRobot*$\}$. The initial state is given by the initial location and colour of each robot and the only accepting states are those that should not be painted. The resulting automaton is shown at Figure 7.

The tile automaton $\mathcal{A}_5$ has three states (`clear`, `occupied`, `painted`), the last one being the goal state if it has to be painted — otherwise all states are accepting. The initial state is either clear or occupied. Which robot acts on the state of the tile is irrelevant, *paint* actions on the given tile transition from clear to painted, and *move* actions transition between clear and occupied depending on whether the given tile is the current or the new one, and *change-colour* actions and actions on other tiles loop. Therefore we can map the original actions to these four: $D^{tile} = \{$*paint, move-to, move-from, other*$\}$. The resulting automaton is shown at Figure 8.

Action costs are not uniform: *change-colour* costs 5, *paint-*$\star$ costs 2, and *move-*$\star$ costs 1 except *move-up* which costs 3. We associate action costs with the robot automaton and our objective to minimize is the sum of each robot's contribution. Let $\mathcal{T}'$ be the set of tiles to paint. We use simple

[4]http://www.plg.inf.uc3m.es/ipc2011-deterministic/DomainsSequential.html

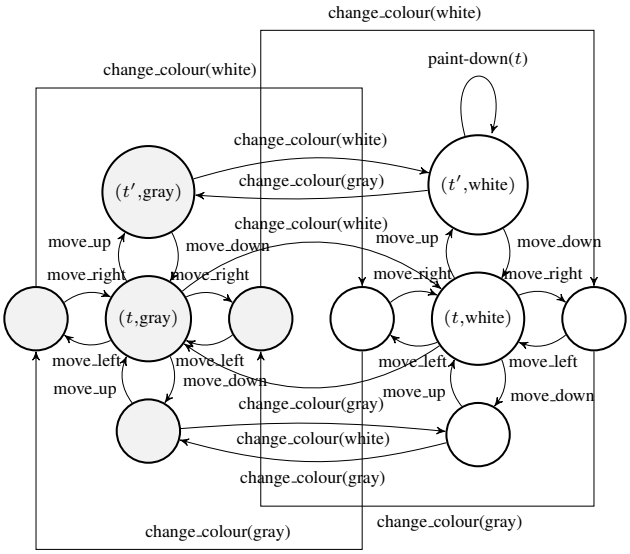

Figure 7: Part of the robot automaton. Loops on *otherRobot* action are not shown. Only one *paint* action is shown, from state (t', white) assuming that tile t needs to be painted white.

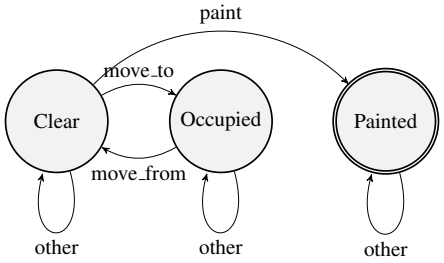

Figure 8: Tile automaton

$\ell_{\min} = |\mathcal{T}'| + \left\lceil \frac{|\mathcal{T}'| - 2n_r}{2} \right\rceil$ (robots need to move in order to paint more than the tiles directly above and below them) and $c_{\min}(\ell) = |\mathcal{T}'| + \ell$ by the same argument as the one for the previous domain. Let $\mathcal{T}$ be the set of tiles and $\mathcal{R}$ the set of robots. Constraints (10) to (19) define the model.

$$\text{minimize } z \tag{10}$$

$$z = \sum_{r \in \mathcal{R}} z_r \tag{11}$$

$$z \geq |\mathcal{T}'| + \ell \tag{12}$$

$$x_i^t = M_t(x_i) \qquad \forall\, 1 \leq i \leq \ell,\ t \in \mathcal{T} \tag{13}$$

$$\textsc{Regular}([x_1^t, \ldots, x_\ell^t], \mathcal{A}_5) \qquad \forall\, t \in \mathcal{T} \tag{14}$$

$$x_i^r = M_r(x_i) \qquad \forall\, 1 \leq i \leq \ell,\ r \in \mathcal{R} \tag{15}$$

$$\textsc{CostRegular}([x_1^r..x_\ell^r], \mathcal{A}_4, C, z_r) \qquad \forall\, r \in \mathcal{R} \tag{16}$$

$$x_i \in D \qquad \forall\, 1 \leq i \leq \ell \tag{17}$$

$$x_i^t \in D^{tile} \qquad \forall\, 1 \leq i \leq \ell,\ t \in \mathcal{T} \tag{18}$$

$$x_i^r \in D^{robot} \qquad \forall\, 1 \leq i \leq \ell,\ r \in \mathcal{R} \tag{19}$$

Empirical Evaluation

Through these experiments we want to show the importance of using effective though still general-purpose heuristics such as counting-based search and how the plans produced by this CP approach compare to those of state-of-the-art planners. The experiments were performed using Intel E5-2683 2.1 GHz processors. The memory limit was set to 8 GB, and the time limit was 30 minutes. The CP models were modeled and solved using IBM ILOG CP version 1.6 with an implementation of the (COST)REGULAR constraint that supplies information for counting-based search.

We used two state-of-the-art planners for comparison. The optimal plans were compared with Delfi1 (Katz et al. 2018), a portfolio planner, and the suboptimal plans with Fast Downward Stone Soup 2018 (FDSS) (Seipp and Röger 2018). They are the winners of the IPC-2018 competition optimal planning and satisficing planning tracks, respectively. These planners are available on the website of IPC-2018[5].

The problem instances were obtained from the *planning.domains* repository, where each planning problem has a unique *problem id*. We used 150 instances for `miconic` (problem ids 268 to 417) and 30 instances for `scanalyzer` (problem ids 2408 to 2437). Initially we considered problems 468 to 487 for `floortile` but these instances are for the most part quite hard to solve and even the state-of-the-art planners often do not produce a plan within our time limit. Therefore we built simpler instances from problems 468, 469, and 470 by only considering the first k tiles to be painted, starting from $k = 3$ to the full set, thus yielding 24 instances.

Comparing Combinatorial Search Heuristics

To search over our CP models, we consider several *general-purpose* combinatorial search heuristics that are not planning-specific: simple lexicographic variable ordering with random value ordering (lexico), reverse lexicographic variable ordering with random value ordering (reverse-lexico), standard smallest domain variable ordering with random value ordering (min-dom), more sophisticated impact-based search (IBS; we use IBM ILOG CP's native version) (Refalo 2004), and two counting-based search variants (maxSD and avgSD) (Pesant, Quimper, and Zanarini 2012). The first variant branches on the variable-value pair appearing with the highest frequency in solutions among all constraints whereas the second one branches on the pair with the highest average frequency taken over the constraints it appears in. For the randomized heuristics, we perform ten runs for each and report the median. Table 1 reports the number of instances for which a plan was found with each heuristic and each planning domain. The left half gives results when the search at each plan length is exhaustive and the right half, when a maximum of ten seconds is allocated per plan length. For reference we provide the number of instances solved to optimality by Delfi1.

We see clearly on every domain that the counting-based search heuristics considered here are far superior to the other ones. One exception is `floortile` without $T_{\max}$: because our lower bound on plan length $\ell_{\min}$ is quite poor, we waste a lot of time proving the infeasibility of insufficient plan lengths. Despite its effectiveness counting-based search, being a heuristic, will sometimes make mistakes and mistakes near the top of the search tree will take a long time to undo in the context of depth-first search. The "maxSD LDS" row shows the impact of replacing depth-first search with limited-discrepancy search (Harvey and Ginsberg 1995) which undoes branching decisions near the top earlier. We see that it never deteriorates performance and in some cases the improvement is significant.

[5]https://ipc2018-classical.bitbucket.io/

What Does Counting-Based Search Do from a Planning Perspective?

Given that counting-based search heuristics appear to be much more effective than the rest, it can be enlightening to examine how they proceed in terms of each planning domain.

Execution traces for `miconic` suggest that all the heuristics from the previous section proceed systematically forward through the plan. Of course this is expected for lexico but not necessarily for the others — a possible explanation is that there may be several planning steps tied for selection and by default we are choosing the earliest one. But the main difference lies in how actions are selected: whereas lexico and min-dom choose the next action arbitrarily among those that are allowed, maxSD and avgSD make smart choices, e.g. starting by boarding each passenger whose origin coincides with the initial floor. While this intuitively makes sense to humans, keep in mind that there is no planning knowledge in our CP model: it simply comes from the calculation that there are more feasible paths of length ℓ in the layered digraph built from the passenger automaton that immediately transition to the next `boarded` state (see Fig. 4).

With `scanalyzer` we start to see a less systematic progression of the plan: the counting-based search heuristics, IBS and even min-dom jump around while fixing plan steps but still tend to proceed in an overall forward manner.

For `floortile` we observe that min-dom returns to a systematically forward progression but again without any guidance for action selection. IBS, maxSD and avgSD do not necessarily proceed as systematically and again the last two make smart choices of actions, e.g. painting the tile currently above a robot or moving one robot out of the way before having another paint a tile. The explanation is likely similar to the one given for `miconic` (see Fig.8).

We believe that an in-depth analysis of the behaviour of counting-based search heuristics on planning problems and of their relationship with existing planning heuristics is a very promising line of research that we intend to investigate further.

Comparing to State-of-the-Art Planners

Table 1 already gave an indication of the performance of our CP approach relative to Delfi1. Solving planning problems to optimality and especially proving it can be challenging. This has motivated the planning community to study algorithms that find a *reasonably good* plan within a *reasonably short* amount of time (coined *satisficing*). Figure 9 compares our CP approach to the state-of-the-art satisficing classical planner FDSS for each of the three planning domains. Each cross in the plot represents an instance. A red cross means that the first plan in the series of FDSS plans which was as good as that of the CP-based planner was strictly better than it — a blue cross means that it had the same cost. If FDSS never finds as good a plan, we use the timeout value (1800 seconds) for FDSS. The CP approach shows excellent performance on `miconic` (though hard to see, there are 29 red crosses and 121 blue crosses) and very good performance on `scanalyzer`. It is not competitive on `floortile` but as we mentioned earlier our lower bound on plan length is particularly weak so we waste a lot of time getting to a feasible plan length.

Computational Efficiency of the Approach

The relative success of this CP approach to classical planning is due in large part to the use of global constraints — and automata-based

	without $T_{\max}$			with $T_{\max} = 10$ secs		
	miconic	scanalyzer	floortile	miconic	scanalyzer	floortile
#instances	150	30	24	150	30	24
lexico	20 (19)	10 (9)	6 (6)	17 (17)	8 (8)	6 (5)
reverse-lexico	20 (19)	10 (9)	6 (6)	17 (17)	8 (8)	4 (4)
min-dom	20 (19)	10 (9)	6 (6)	17 (17)	8 (8)	6 (5)
IBS	21 (21)	10 (9)	6 (6)	15 (15)	8 (8)	5 (4)
avgSD	63 (44)	15 (14)	5 (5)	150 (43)	22 (14)	12 (5)
maxSD	65 (43)	13 (12)	6 (6)	150 (42)	20 (11)	11 (5)
maxSD LDS	85 (68)	14 (13)	6 (6)	150 (52)	21 (12)	14 (8)
Delfi1	(141)	(15)	(24)			

Table 1: Number of instances for which a plan was obtained using each branching heuristic. The number of instances for which an optimal plan was found appears in parentheses. The computational time limit is 30 minutes per instance.

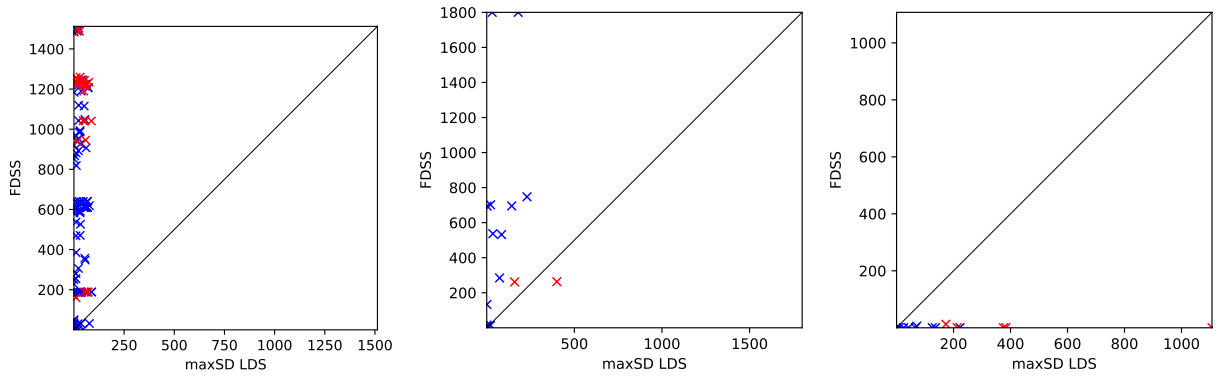

Figure 9: Comparing runtimes of satisficing planner FDSS and of our CP model ($T_{\max} = 10$ secs) with counting-based search heuristic maxSD and limited-discrepancy tree search on each planning domain (from left to right: `miconic`, `scanalyzer`, `floortile`). Each cross represents an instance. Crosses are blue when a plan of same cost was found and red when FDSS found a better plan.

constraints in particular — because they offer a fairly direct factored representation of the planning problem into well-established CP components inside of which we already have algorithms to shrink the search space and guide its exploration. The search space reduction had already been exploited in previous work on planning as CP but the guidance through counting-based search heuristics is something new here and our experiments indicate that their impact on performance is very significant.

Another contribution to efficiency is the mapping of the parameterized actions from the original PDDL description into a reduced set either because some of these parameters are already reflected in the states of our automata or because some actions are considered equivalent from the perspective of a particular planning object and hence can be merged into one. This simplifies the description of each automaton and thus reduces the size of the layered digraphs built by the automata-based constraints, which has a definite impact on runtime.

One aspect that could be improved upon is plan-length selection. Our detailed empirical results suggest that once a feasible plan length is reached, our CP models guided by counting-based search quickly find a plan, sometimes without any backtracking. A better strategy than simply increasing plan length from a straightforward lower bound could lead to a significant improvement in performance.

Conclusion

We provided empirical evidence that constraint solvers are already well equipped to solve classical planning problems, even without injecting planning knowledge in the form of specialized techniques, and that they can even be competitive with state-of-the-art planners on some domains. The experiments showed that some general-purpose search heuristics developed in constraint programming work well on classical planning problems but that using an effective heuristic is crucial. All models and techniques we developed are generic, *planning-independent*, and can be applied to other planning domains: we believe that the way we derived our model for each of the three domains is very similar and outlines a systematic approach to modeling classical planning problems in CP. If an additional modeling standard for planning problems, based on automata, were adopted it would make it much easier for CP technology to be used in AI planning.

Acknowledgements

We thank the anonymous reviewers for their insightful comments and suggestions that helped us improve this paper. Financial support for this research was provided by NSERC Discovery Grants 218028/2017 and RGPIN-2016-05953 and an IVADO postdoctoral scholarship.

References

Barták, R., and Toropila, D. 2008. Reformulating constraint models for classical planning. In Wilson, D., and Lane, H. C., eds., *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA*, 525–530. AAAI Press.

Beck, J. C.; Magazzeni, D.; Röger, G.; and Hoeve, W.-J. V. 2018. Planning and Operations Research (Dagstuhl Seminar 18071). *Dagstuhl Reports* 8(2):26–63.

Bessière, C., and Régin, J. 1997. Arc consistency for general constraint networks: Preliminary results. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, 398–404. Morgan Kaufmann.

Demassey, S.; Pesant, G.; and Rousseau, L. 2006. A cost-regular based hybrid column generation approach. *Constraints* 11(4):315–333.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artif. Intell.* 132(2):151–182.

Ghooshchi, N. G.; Namazi, M.; Newton, M. A. H.; and Sattar, A. 2017. Encoding domain transitions for constraint-based planning. *J. Artif. Intell. Res.* 58:905–966.

Gregory, P.; Long, D.; and Fox, M. 2010. Constraint based planning with composable substate graphs. In Coelho, H.; Studer, R.; and Wooldridge, M. J., eds., *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 453–458. IOS Press.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, 607–615. Morgan Kaufmann.

Helmert, M., and Lasinger, H. 2010. The scanalyzer domain: Greenhouse logistics as a planning problem. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 234–237. AAAI.

Judge, M., and Long, D. 2011. Heuristically Guided Constraint Satisfaction for Planning. In *Proceedings of the 29th Workshop of the UK Planning and Scheduling Special Interest Group*.

Katz, M.; Sohrabi, S.; Samulowitz, H.; and Sievers, S. 2018. Delfi: Online planner selection for cost-optimal planning. IPC-2018 planner abstracts. Available at https://ipc2018-classical.bitbucket.io/planner-abstracts/teams_23_24.pdf.

Koehler, J., and Schuster, K. 2000. Elevator control as a planning problem. In Chien, S. A.; Kambhampati, S.; and Knoblock, C. A., eds., *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, 331–338. AAAI.

Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a CSP. In Gottlob, G., and Walsh, T., eds., *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 954–960. Morgan Kaufmann.

Pesant, G., and Zanarini, A. 2011. Recovering indirect solution densities for counting-based branching heuristics. In Achterberg, T., and Beck, J. C., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems - 8th International Conference, CPAIOR 2011, Berlin, Germany, May 23-27, 2011. Proceedings*, volume 6697 of *Lecture Notes in Computer Science*, 170–175. Springer.

Pesant, G.; Quimper, C.; and Zanarini, A. 2012. Counting-based search: Branching heuristics for constraint satisfaction problems. *J. Artif. Intell. Res.* 43:173–210.

Pesant, G. 2004. A regular language membership constraint for finite sequences of variables. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, 482–295.

Refalo, P. 2004. Impact-based search strategies for constraint programming. In Wallace, M., ed., *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, 557–571. Springer.

Seipp, J., and Röger, G. 2018. Fast downward stone soup 2018. IPC-2018 planner abstracts. Available at https://ipc2018-classical.bitbucket.io/planner-abstracts/team45.pdf.

van Beek, P., and Chen, X. 1999. Cplan: A constraint programming approach to planning. In Hendler, J., and Subramanian, D., eds., *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, 585–590. AAAI Press / The MIT Press.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artif. Intell.* 170(3):298–335.

Zanarini, A., and Pesant, G. 2009. Solution counting algorithms for constraint-centered search heuristics. *Constraints* 14(3):392–413.

Zanarini, A.; Pesant, G.; and Milano, M. 2006. Planning with Soft Regular Constraints. ICAPS Workshop on Preferences and Soft Constraints in Planning. Available at `https://www.polymtl.ca/labo-quosseca/en/publications`.