Improvements to CPCES

Xiaodi Zhang, Alban Grastien

School of Computing, The Australian National University
xiaodi.zhang@anu.edu.au, alban.grastien@anu.edu.au

Abstract

This paper introduces three improvements to the conformant planner CPCES, which continuously searches candidate plans and counter-examples against the current candidate plan until a valid plan (no counter-example exists) is found. First, we identify and merge equivalent PDDL facts to accelerate candidate plan generation. Second, we warm-start CPCES by generating multiple carefully selected counter-examples at the beginning of the procedure, which reduces the number of calls to the classical planner. Third, we investigate the use of Fast Downward (FD) as the candidate plan generator; in particular, we propose an incremental procedure to generate the SAS+ file used by FD. Our experimental results show significant improvements for each technique.

1 Introduction

Conformant planning is the problem of generating a plan despite uncertainty on the initial state or the actions' effects, and no observation during plan execution (Smith and Weld 1998). In this paper, we concentrate on *deterministic* conformant planning in which the uncertainty is only in the initial state: the plan should be valid for all possible initial states.

Conformant planning is EXPSPACE-COMPLETE (Haslum and Jonsson 1999), and existing planners can only solve problems that are tiny compared to other planning frameworks. It is therefore crucial to consider some of the engineering aspects of the implementation. In this paper, we present three improvements to CPCES, a state-of-the-art conformant planner.

CPCES (Grastien and Scala 2017, 2020) is a conformant planner that uses two components. Given a candidate plan, the first component searches for an initial state (*sample*) in which the plan is invalid; this subproblem is solved by a SAT solver. Given a sample set, the second component computes a candidate plan that is valid for all these states; this subproblem is reduced to a classical planning problem with a multi-interpretation PDDL file (McDermott et al. 1998). CPCES collects the samples computed throughout the procedure and generates new candidate plans from this sample set until a conformant plan is found. In this way, CPCES does not need to generate all initial states.

Projection link: https://github.com/xiaodizhang8/ICPCES-2023.git

In this paper, we propose three improvements to CPCES. First, we improve the PDDL representation of the classical planning subproblem used to generate the candidate plan. The planning subproblems build on top of a multi-interpretation representation of the environment, each interpretation corresponding to one sample. However, some PDDL facts have the same evaluation across all interpretations. We identify these facts, merge them in the PDDL file, and thereby simplify the planning subproblem.

CPCES normally starts with an empty sample set. Our second improvement is a warm start to CPCES where the sample set is initialised with some initial states. We study how the choice of the initial sample set affects performance.

CPCES can be used with any classical planner. Traditionally, Fast Forward (FF) (Hoffmann and Nebel 2001) offers the best performance compared to, e.g., madagascar (Rintanen 2012) and Fast Downward (FD) (Helmert 2006). We want to improve performance for FD as this planner offers more options than FF and can compute optimal plans. Our third contribution is a better integration of FD in CPCES. Early experiments have shown us that the translation from PDDL to SAS+ (the first step of FD) is time consuming and leads to poor SAS+ representations. However, we recognise that the classical planning problems have a structure that can be exploited. Each problem contains the list of interpretations of the previous iteration augmented by one. We propose to isolate this new interpretation, translate it independently to SAS+, and add this translation to the existing translation. This is not only faster, as the translation becomes trivial; it also leads to better SAS+ translations.

This paper is structured as follows. We first give the background of this paper. Then, we introduce the improvement of classical planning problem representation. Next, we show how to warm-start CPCES to find a valid plan quickly. Finally, we present how to exploit the structure of the classical planning problems in order to accelerate its translation to SAS+ and integrate FD in CPCES. Each improvement introduced in this paper is validated by experiment results.

2 Background

Conformant Planning

Given a set of *facts* F, a *state* is modelled as the subset of facts $s \subseteq F$ that are true in the state. Given a set F, we write

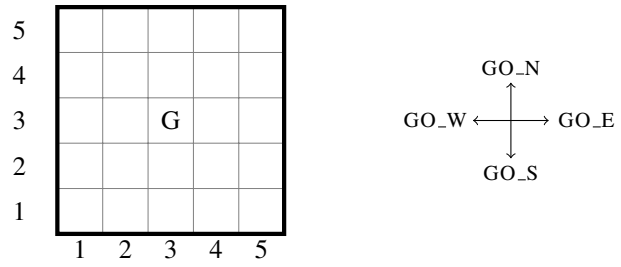

Figure 1: Graphical representation of a conformant planning problem. A robot is standing in this 5×5 grid surrounded by walls. Its initial location is unknown. The goal is to move the robot to "G". One valid plan is GO_E $\times$ 4, GO_S $\times$ 4, GO_N $\times$ 2, GO_W $\times$ 2.

$\mathcal{L}(F)$ the set of propositional formulas over F, where F is interpreted as a set of propositional variables. A *deterministic conformant planning* problem $P = \langle F, N, A, I, G\rangle$ is defined as follows:

- F is the finite set of *facts*;
- N is a set of *action names*, also called *actions*;
- $A : N \to \mathcal{L}(F) \times 2^{\mathcal{L}(F)\times 2^F \times 2^F}$ is a function that returns a description of each action such that $A(a) = \langle pre(a), \mathit{coneff}(a)\rangle$ and $\mathit{coneff}(a) = \{\langle c_1, \mathit{eff}^+_1, \mathit{eff}^-_1\rangle, \ldots, \langle c_k, \mathit{eff}^+_k, \mathit{eff}^-_k\rangle\}$;
- $I \in \mathcal{L}(F)$ is the *initial condition*; and
- $G \in \mathcal{L}(F)$ is the *goal condition*.

A state s is *initial* if it satisfies the initial condition: $s \models I$; similarly, a state is a *goal state* if it satisfies the goal condition. Applying action a in state s yields the *positive effects* $\mathit{eff}^+(s, a)$ defined by

$$\mathit{eff}^+(s,a) = \bigcup_{\substack{\langle c, \mathit{eff}^+, \mathit{eff}^-\rangle \in \mathit{coneff}(a)\\ s \models c}} \mathit{eff}^+.$$

The application also yields *negative effects* $\mathit{eff}^-(s, a)$ defined similarly. The state reached when applying action a in state s is $s[a] = s \setminus \mathit{eff}^-(s,a) \cup \mathit{eff}^+(s,a)$. Action a is *applicable* in state s if the following two conditions hold:

1. state s satisfies the action's precondition: $s \models pre(a)$;
2. the effects do not conflict: $\mathit{eff}^+(s,a) \cap \mathit{eff}^-(s,a) = \emptyset$.

A *plan* is a sequence of actions $\pi = a_1, \ldots, a_k$. *Applying* this plan in state s_0 leads to the sequence of states $s_1, \ldots, s_k$ where $s_i = s_{i-1}[a_i]$ for each i. We write $s_k = s[\pi]$. The plan is *applicable* in state s_0 if each action a_i is applicable in s_{i-1}; it is *valid* in s_0 if it is applicable in s_0 and s_k is a goal state. A *solution* to the conformant planning problem (a *conformant plan*) is a plan that is valid in all initial states. We write $\Pi(P) \subseteq N^*$ the set of conformant plans of P.

A conformant planning problem is a *classical* planning problem if there is only one initial state.

Algorithm 1: The conformant planner CPCES.

```
1: input: conformant planning problem P
2: output: a conformant plan, or no plan
3: B := ∅ // empty sample set
4: π := ϵ // empty plan
5: loop
6:    q := generate-counter-example(P, π)
7:    if there is no such q then return π
8:    B := B ∪ {q}
9:    π := produce-candidate-plan(P, B)
10:   if there is no such π then return no plan
11: end loop
```

Example 1 *In Figure 1, a robot is standing in a 5×5 grid, but its exact location is unknown. The grid is surrounded by walls to prevent the robot from moving outside. If the robot hits the wall, it will not move but just stand still. The robot can execute four actions:* GO_N, GO_E, GO_W, GO_S, *which represent moves to North, East, West, and South, respectively. The goal is to get the robot to location (3,3), denoted as "G". A conformant plan is* GO_E $\times$ *4,* GO_S $\times$ *4,* GO_N $\times$ *2,* GO_W $\times$ *2.*

CPCES

CPCES (Grastien and Scala 2020) is a conformant planner summarised in Algorithm 1. CPCES searches for a conformant plan by continuously computing candidate plans and counter-examples to these candidates. A counter-example is an initial state for which the candidate plan is not valid. CPCES generates a counter-example to the current candidate plan on Line 6 and stores it in the *sample set* B. Then, CPCES produces a candidate plan valid for all states in B (Line 9). Adding the new counter-example in B guarantees that CPCES will never produce the same (invalid) candidate plan more than once.

Relevant to this paper is how the subproblem of generating a new candidate plan is reduced to classical planning. This is done by using a multi-interpretation of the conformant planning problem where each interpretation is linked to one sample from the sample set. So, each fact $f \in F$ is cloned into a set of facts $f\text{-}int1, \ldots, f\text{-}intn$ where each fact $f\text{-}inti$ represents the value of fact f assuming the initial state was the sample s_i. The definitions of the actions, initial condition, and goal condition are updated appropriately. This reduction yields a classical planning problem which can be solved with an off-the-shelf classical planner.

3 Related Work

Conformant planning is a research topic that attracted a significant amount of research in the last two decades, and we will not provide a complete description of the existing work here. The problem has been looked at from very different perspectives. Some are based on exploiting compact representations for the belief tracking such as BDDs (Cimatti and Roveri 2000) or CNFs (To, Pontelli, and Son 2011). One of the difficulties of these approaches lies in coming up with good ways to target the particular structure of the problem

at hand. To overcome this limitation, other researchers have proposed a more explicit approach for the exploration of the belief states which could make a more direct use of the power of heuristic search (Bonet and Geffner 2000). The former of this family of work is due to Hoffmann and Brafman (2006), with the Conformant FF planner, an extension of FF (Hoffmann and Nebel 2001) that reasons over the belief entailed by the prefix under exploration. A SAT solver is used to check entailment of preconditions and goals.

Another branch of research has instead focused on the exploitation of decompositions that can be automatically extracted from the structure of the problem. Notions as tags and width have shown a useful characterisation of such decompositions, and have led to powerful reductions to classical planning. These decompositions have been studied extensively by Palacios and Geffner (2009) and Albore, Palacios, and Geffner (2010). In particular, in a close but successive work (Albore, Ramirez, and Geffner 2011) the authors managed to combine the Conformant-FF basic search over the belief with a novel tag-based heuristic obtained from an unsound-but-complete reduction to classical planning.

Other approaches use a sampling based mechanism as CPCES does. The fragment planner (Kurien, Nayak, and Smith 2002) tries to find a conformant plan by combining plans resulting from solving each initial state independently; such initial states are randomly sampled. The authors investigate different ways of performing this search, yet none guarantees a systematic reduction of the initial states that is able to exploit the structure of the problem; as noted by Nguyen et al. (2012), the fragment planner does not scale well. With the aim of overcoming the fragment planner limitations, Nguyen et al. (2012) propose the so called generate-and-complete approach. The idea is to find a plan for a subproblem $P(s_0)$ of the conformant planning problem using a classical planner, try to repair it to account for other initial states, and if that does not work explore another classical planning solution for $P(s_0)$.

4 Merging Certain-Facts

Our first improvement relies on the notion of a *certain-fact*, which is a fact whose value is always known during plan execution. In other words, such a fact holds in the state reached by applying any (applicable) sequence of actions from an initial state iff it holds in all states reached by applying this sequence from any initial state.

Definition 1 *A fact $f \in F$ is a* certain-fact *if for any plan π applicable in all initial states and for any pair (s, s') of initial states, the property $f \in s[\pi] \Leftrightarrow f \in s'[\pi]$ holds.*

Deciding if a fact is a certain-fact is EXPSPACE-COMPLETE (it requires being able to decide if a specific action is ever applicable). Therefore, we use an approximate procedure based on the structural notion of context (Palacios and Geffner 2009) which we recall here.

A *subgoal* ϕ is a conjunct of the goal condition or of the precondition of some action.[1] A fact f_1 *depends* on another fact f_2 if there is a conditional effect $\langle c, \mathit{eff}^+, \mathit{eff}^- \rangle$ such that f_1 is in $\mathit{eff}^+ \cup \mathit{eff}^-$ and f_2 appears in condition c. The *context* $ctx(\phi)$ of a subgoal ϕ is the set of facts in the subgoal, alongside the facts they depend on, transitively.

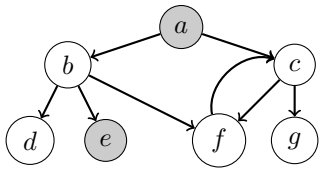

Figure 2: Illustrating the certain-facts: edges represent dependency and shaded nodes (a and e are facts that are initial unknown). Here, c, d, f, and g are known certain-facts.

The *context graph* is a directed graph whose vertices are the facts F and an edge from f to f' indicates that there exists an action with a conditional effect $\langle c, \mathit{eff}^+, \mathit{eff}^- \rangle$ where $f \in \mathit{eff}^+ \cup \mathit{eff}^-$ is an effect and f' appears in c.

Given a plan π, the *belief* is the set of states that the system could be in, and is formally defined as $\mathcal{B} = \{s[\pi] \mid s \models I\}$. We say that a fact is *known* in a belief if all states agree on this fact, i.e., $\forall s, s' \in \mathcal{B}.\ f \in s \Leftrightarrow f \in s'$.

Lemma 1 *Let f be a fact such that all the facts it depends on (including itself) are known in the initial belief. Then f is a certain-fact.*

Proof sketch: Let F' be the set of facts that f depends on. We note that if f' is a fact from F', all facts that f' depends on are in F'. We prove Lemma 1 by showing, by induction over the plans, that all facts in F' are certain-facts.

Base case ($\pi = \varepsilon$): This is the assumption of the lemma, that all facts in F' are known in the initial belief.

Inductive step: Assume all facts F' are known in the belief $\mathcal{B}_i$ reached by applying $\pi = a_1, \ldots, a_i$.

Are all facts of F' known in the belief $\mathcal{B}_{i+1}$ reached by applying $a_1, \ldots, a_i, a_{i+1}$? By contradiction, assume that there is a fact $f' \in F'$ that is not known at step $i+1$. Then, there are two states s_1 and s_2 from $\mathcal{B}_{i+1}$ where $f' \in s_1$ and $f' \notin s_2$ hold. Let s'_1 and s'_2 be the two states from $\mathcal{B}_i$ such that $s_1 = s'_1[a_{i+1}]$ and $s_2 = s'_2[a_{i+1}]$. Since f' is known in $\mathcal{B}_i$ (induction hypothesis), f' is either in both s'_1 and s'_2, or neither. Wlog, let us assume $f' \in s'_1 \cap s'_2$. Since f' is not in s_2, a conditional effect $\langle c, \mathit{eff}^+, \mathit{eff}^- \rangle$ of a_{i+1} must have triggered from state s'_2. However, we note that condition c only involves facts from F' which are all known in $\mathcal{B}_i$. Therefore, the conditional effect also applies to s'_1, and f' should not be in s_1, which contradicts the assumption. This concludes the inductive step and, hence, the lemma.

Example 2 *This is illustrated in the context graph of Figure 2. The shaded nodes represent facts that are initially unknown. Facts c, d, f, and g are provably certain. Fact b depends on e which is initially unknown; it might be a certain-fact but this cannot be proven with Lemma 1.*

CPCES translates a conformant planning problem to multiple parallel subproblems and transforms subproblems into a classical planning problem with a multi-interpretation

[1] We interpret each formula as a conjunction; for instance, the formula $\varphi = a \vee (b \wedge c)$ contains one conjunct which is precisely φ.

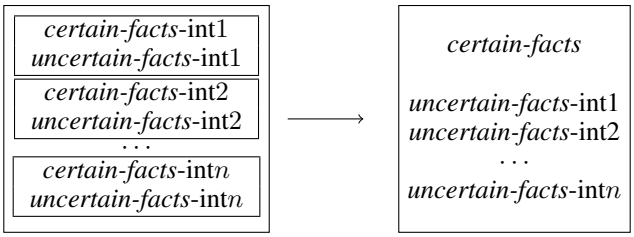

Figure 3: The certain-facts in the interpretation PDDL file can be merged and shared across all the interpretations.

PDDL file. This is illustrated on the left side of Figure 3. Because the different interpretations of the same certain-facts are logically equivalent, we only need one copy of the certain-facts for all interpretations. We need to modify the definitions of the initial state, goal, and actions accordingly. We expect that the PDDL file will be more compact. We also expect the planner to perform better (for instance, the heuristic used by a heuristic-search planner should be more accurate).

Experimental Results

We compared two settings: merging certain-facts (MF) and not merging certain-facts (NMF). We measure the amount of time needed to find a valid plan. We expect MF performs much better than NMF.

We ran experiments over the set of benchmarks from Zhang, Grastien, and Scala (2020). The benchmark set contains 7 domains: DISPOSE, ONEDISPOSE, BLOCKWORLD, LOOK-GRAB, RAOSKEYS, BOMB, and COINS. Experiments were run on Ubuntu virtual machine with 8GB memory and 1 processor on Apple M1 chip Macbook Pro. Timeout was set to 1,800 seconds (30 mins). As CPCES includes non-deterministic steps, each instance was solved 19 times and we report the median one. To keep the search time and number of iterations stable, we used SUPERB (Zhang, Grastien, and Scala 2020). SUPERB includes an additional inner loop when computing counter-examples that tries to generate counter-examples that include as many tags as possible (so-called *superior counter-examples*) as these counter-examples provide more information to find better candidate plans and ultimately reduce the number of CPCES iterations.

Table 2 shows results for some selected problems. The complementary material contains the full experimental results. Comparing columns NMF and MF for Total Time, we see that merging facts is beneficial for most non-trivial problems (those that take at least one second). For example, although both MF and NMF solve DISPOSE p-8-2 in 65 iterations, MF is about 60% faster than NMF. ONEDISPOSE and LOOK-GRAB show similar results. For some problems that have no certain-fact (certain-facts ratio is 0), MF shows a great improvement as well. For example, solving BOMB p100-5, MF is 40% faster than NMF. This is because more than 99% of facts are constants, which are merged too.

Surprisingly, the instance p-3-3 of ONEDISPOSE shows opposite results. We do not have a good explanation for this phenomenon. We note that the number of CPCES iterations for this instance is significantly higher under MF. We presume that the heuristic must be driving FF towards bad candidate plans. Without more similar instances, it is impossible to draw conclusions.

5 Warm-Starting CPCES

CPCES incrementally samples the states from the initial set until the classical planner generates a valid plan. CPCES starts with an empty sample set (Line 3 of Algorithm 1); we refer to this as a *cold start*. Unfortunately, this approach requires a large number of iterations before CPCES starts generating good candidates. We study here a *warm start* strategy in which the sample set is already populated with a number of initial states. The problem is then to decide which initial states to choose.

We know that the sample sets of a given cardinality are not equally good. Palacios and Geffner (2009) have proven that the important feature of an initial state is its set of *tags*: for a given context (subset of facts), a tag of a state is the intersection of the state with the context. It is possible to compute a sample set that will cover all tags of the initial set: this would guarantee the validity of the first candidate plan. The number of initial states thus sampled might be too large however, and all tags are not always necessary. The problem of Figure 1 for instance, features ten tags: x_i (the robot is in column i) and y_i (in row i) for $i \in \{1, \ldots, 5\}$; however, when equipped with a sample set that covers the four tags x_j, y_j for $j \in \{1, 5\}$, CPCES will always return a valid plan at the next iteration.

By construction, CPCES always computes "useful" initial states: each new initial state is a counter-example of the current candidate plan and, therefore, guarantees that a different plan will be generated at the next iteration. Similarly, our objective is to select a "good" set of initial states or, equivalently, to guess which tags need to be covered.

Our criterion is based on the context graph as defined in Section 4. Remember that if there is an edge from f to f', then f depends on f', i.e., an action exists such that the current value of f' might influence the value of f after the application of this action. If the shortest path from f to f' is of length two, then at least two actions are needed before the value of f' may affect the value of f. Our intuition is that the initial states that are the most difficult to deal with are those that include facts that have the longest shortest paths.

Definition 2 *The* distance *from fact f to fact f' is the minimal number of steps from f to f' in the context graph. If there is no such path, the distance is assumed to be zero. The* score *of a fact f is the largest distance from this fact to any other fact.*

Example 3 *This is illustrated on Figure 4 that represents the contexts of the planning problem of Figure 1. This graph shows that x_1 depends directly on x_2 but indirectly on the other facts. The d value then represents the number of links to the furthest fact. In other words, the d value is the score. 4 in the case of x_1 and x_5, 2 for x_3. Following our intuition, we*

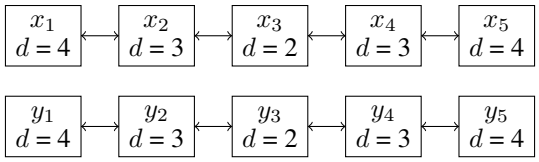

Figure 4: Two context graphs with the score of the facts in Example 3. x and y are two facts refer to the abscissa and the ordinate of the robot. d is the score

Algorithm 2: Generating important samples

input: conformant planning problem P
output: a set of important samples
$S := \texttt{compute-scores}(P)$
$F := \texttt{pickup-important-facts}(P, S)$
$B := \emptyset$
loop
  $q := \texttt{generate-important-sample}(B, F)$
  if there is no such q **then return** B
  $B := B \cup \{q\}$
end loop

consider that we should prioritise initial states that include tags x_1 or x_5.

An uncertain-fact is called an *important fact* if it maximises the score within some context graphs. If all uncertain-facts of an initial state are important, we say that the state is *important*.

Warm-Starting

Warm-starting CPCES means initialising the sample set with important samples. Here, we require that the sample set be the smallest: each important fact can appear only once in the sample set. The warm-starting procedure is presented in Algorithm 2: we first compute the score of each fact and extract the important facts. Then, using an SMT solver (De Moura and Bjørner 2008), we repeatedly search for initial states that contain new important facts until no new states are found. Since we never compute two initial states that feature the same important fact, this procedure eventually terminates.

Experimental Results

The benchmarks and the experimental setup are the same as Section 4. The results are summarised in Table 2. The performance of the warm starting method is represented by column WS and incorporates the certain-fact-merging method; therefore, it should be compared to column MF. Ideally, the important samples we get are sufficient for CPCES to obtain a valid solution immediately; when this happens, CPCES only needs two steps (the first step finds a valid candidate and the second step finds no counter-example to this candidate) to end the algorithm.

We see that warm-started CPCES is often able to find solutions very quickly, generally reducing the search time by several orders of magnitude. For instance, the total time for DISPOSE p-8-3 drops from 204s to 10s; ONEDISPOSE p-3-3 from 235s to less than a second.

We note however that, in domains such as LOOK-GRAB, warm-starting actually harms performance. We believe that this is because the sample set is too large. Indeed, for instance p-8-3-3, the sample set has size 64 while cold-start methods only generate 7 or 8 samples.

6 Integrating FD Into CPCES

CPCES can be used with any classical planner that is able to read PDDL files and output a plan in an appropriate format. In particular, this is true of `fast-downward` (FD) (Helmert 2006). FD is an appealing planner because it offers a wide range of heuristics and planning procedures. However, early experiments have shown that FD is significantly slower than FF. This is illustrated on Table 2 when comparing column NMF (FF) and column FD (FD). We see that, with one exception, the implementation that uses FF consistently performs better, and sometimes several orders of magnitude better. Using FD based on LAMA-2011 (Richter, Westphal, and Helmert 2011) has almost the same results.

We identified that one important issue for FD is the translation of the planning problem from PDDL to SAS+, a procedure that is repeated at every iteration of CPCES. Therefore, we propose to build the SAS+ file incrementally. In order to explain the procedure, we first introduce SAS+.

SAS+

The problem definition given in Sections 4 and 5 matches the PDDL representation in which the basic element is a *fact*. Instead, FD uses a multi-valued encoding such as SAS+ (Bäckström and Nebel 1995; Helmert 2009).

The basic element in a multi-valued encoding is the *state variable*. Each state variable v has a domain D_v. A state is defined as a complete assignment of the state variables. An atomic effect $(v = \nu)$, which assigns the variable v to ν, then corresponds to a positive effect $(v = \nu)$ and a list of negative effects $(v = \nu')$ for all other value $\nu' \in D_v \setminus \{\nu\}$.

Translating a PDDL representation to a SAS+ representation generally involves identifying subsets of pairwise mutually exclusive facts F' (Helmert 2006). For each such subset, a variable v is created with domain $D_v = F' \cup \{\bot\}$: variable v is assigned to $f \in F'$ in a SAS+ state iff fact f would appear in the equivalent PDDL state: $s \cap F' = \{f\}$. Symbol $\bot$ represents a situation in which none of the facts from F' appears in the state: $s \cap F' = \emptyset$.

One advantage of multi-valued encoding is that state representation is more compact (Bäckström and Nebel 1995); it also enables richer heuristic functions (Geffner 2007; Richter, Helmert, and Westphal 2008; Richter and Westphal 2010).

Incremental SAS+

We have seen that CPCES can be used with any complete classical planner able to parse PDDL files and to output a valid plan if it exists, including FD. We hypothesised that the poor performance of FD compared to FF was due to its translation to SAS+. Specifically, we see two issues. First,

the problems that need to be translated are large and include the `forall` PDDL construct which is hard to handle. This is illustrated in Table 1. In this experiment, we replaced the `forall` with an explicit enumeration of the interpretations in the PDDL file. We see that the version without `forall` requires up to 30 times fewer variables. With the current representation, the `FD` search engine is unable to fully exploit its capabilities. Second, a translation from PDDL to SAS+ is performed at each iteration of `CPCES`, which has a non-negligible cost.

We acknowledge here that the sequence of classical planning problems that `CPCES` sends to the classical planner has a peculiar structure that can be exploited. Indeed, each problem is an "increment" of the previous problem in which the set of PDDL facts is increased while the set of actions, as well as their relation with the existing facts, remains unchanged. We formalise this intuition now.

Definition 3 *Two planning problems P_1 and P_2, where $P_i = \langle F_i, N_i, A_i, I_i, G_i\rangle$ for $i \in \{1, 2\}$, are* independent *if their sets of action names are identical and their sets of facts are disjoint: $(N_1 = N_2) \wedge (F_1 \cap F_2 = \emptyset)$.*

We also propose the definition of merge, which joins two problems with the same set of action names.

Definition 4 *Let P_1 and P_2 be two planning problems. where $P_i = \langle F_i, N_i, A_i, I_i, G_i\rangle$ for $i \in \{1, 2\}$. The* merge *of P_1 and P_2, denoted $P_1 \oplus P_2$, is a problem $P = \langle F, N, A, I, G\rangle$ defined by:*

- $F = F_1 \cup F_2$; $\quad I = I_1 \wedge I_2$; $\quad G = G_1 \wedge G_2$;
- $N = N_1 = N_2$; *and*
- *A is such that $A(a) = \langle pre_1(a) \wedge pre_2(a), coneff_1(a) \cup coneff_2(a)\rangle$ for all $a \in N$.*

Definition 4 translates naturally to the SAS+ notation, and we therefore use the operator $\oplus$ for SAS+ problems as well.

We also use the notation $P \oplus P_\Delta$ and call P_Δ an *increment* to the *original problem* P. A problem increment is therefore a refinement of an existing planning problem that defines new facts relevant to the planning task and specifies how the actions interact with these facts. The increment also satisfies the property that it does not directly interact with the existing facts.

The benefit of the notions of merge and increment is that it allows for a simple translation to SAS+ when a translation of the original problem already exists. This is summarised in the following lemma:

Lemma 2 *Let P be a PDDL planning problem, and let P_Δ be an increment to P. Let S and S_Δ be SAS+ representations of P and P_Δ respectively such that*

- $\Pi(S) = \Pi(P)$;
- $\Pi(S_\Delta) = \Pi(P_\Delta)$; *and*
- *the sets of variables of S and S_Δ are disjoint.*

Then $S \oplus S_\Delta$ is a SAS+ representation of $P \oplus P_\Delta$ such that $\Pi(S \oplus S_\Delta) = \Pi(P \oplus P_\Delta)$.

Proof sketch: The proof relies on the property that when two independent planning problems are merged, be they modelled in PDDL or SAS+, the valid solutions of the resulting problem are the intersection of the solutions of the original problems. Hence:

$$\begin{aligned}\Pi(S \oplus S_\Delta) &= \Pi(S) \cap \Pi(S_\Delta)\\ &= \Pi(P) \cap \Pi(P_\Delta)\\ &= \Pi(P \oplus P_\Delta).\end{aligned}$$

From Lemma 2, we now have a procedure for computing efficiently SAS+ representations of the classical planning problems produced by `CPCES`. At each iteration, the classical planning problem $P_{1\ldots i}$ is defined as the merge of all the classical planning problems P_1, P_2, … to P_i corresponding to the i interpretations (or counter-examples) generated so far throughout the procedure. $P_{1\ldots i}$ is also the merge of $P_{1\ldots i-1}$ and P_i. The SAS+ representation of $P_{1\ldots i}$ can therefore be obtained by merging the SAS+ representation of $P_{1\ldots i-1}$ (computed at the previous `CPCES` iteration) with a SAS+ representation of P_i. This is illustrated on Figure 5 where the vertical operation that translates from $P_{1\ldots i}$ to $S_{1\ldots i}$ is replaced with the decomposition (to the right), the vertical translation of each part, and the merge (to the left).

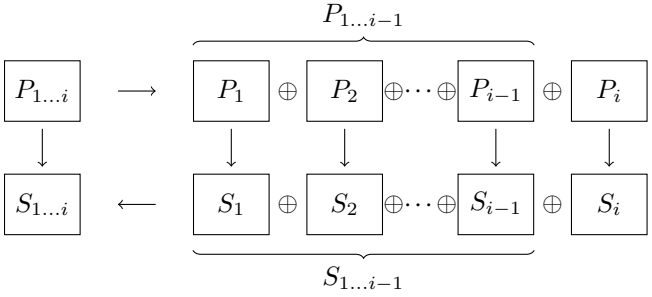

Figure 5: Graphical representation of the decomposition proposed in this paper: the classical planning problem $P_{1\ldots i}$ that `CPCES(FF)` needs solved at the ith iteration can be decomposed into i planning problems P_j. Each of these problems can be translated into its own SAS+ representation, S_j, and these representations merged into a SAS+ representation $S_{1\ldots i}$ equivalent to $P_{1\ldots i}$.

Algorithm 3: Incremental generation of candidate plan.

1: **method**: `produce-candidate-plan`(P, B)
2: **input**: conformant planning problem $P = \langle F, N, A, I, G\rangle$
3: **input**: belief B
4: **output**: a candidate plan, or **no plan**
5: **static**: a map $S : 2^F \to$ SAS_FILES
6: **if** $B = \emptyset$ **then return** ε
7: **for all** $q \in B$ **do**
8: **if** $S(q)$ is undefined **then**
9: $S(q) :=$ `generate-sas-plus`$(\langle F, N, A, q, G\rangle)$
10: **end if**
11: **end for**
12: $SAS := \bigoplus_{q \in B} S(q)$
13: **return** `FD`(SAS)

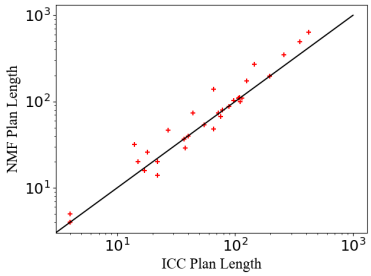

Figure 6: Comparison of plan length between ICC and NMF.

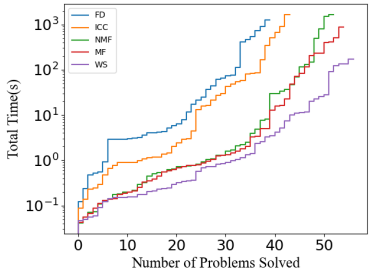

Figure 7: Number of problems solved by each algorithm.

Domain	Instance	With	Without
BLOCKWORLD	p02	115	42
BOMB	p20-5	625	117
COINS	p15	903	432
DISPOSE	p4-2	641	33
LOOK-GRAB	p4-2-2	442	14
ONEDISPOSE	p3-2	500	76
RAOSKEYS	p2	39	18

Table 1: Number of variables in the SAS+ problem with and without `forall`.

Our approach is summarised in Algo. 3 which implements the method `produce-candidate-plan` (Line 9) from `CPCES` (Algo. 1). This algorithm assumes a static map/dictionary that keeps track of the SAS translation associated with each state that it considered in the belief.[2] When new states are available (i.e., $S(q)$ is undefined), a SAS+ representation of this state is computed. Then, Line 12, the SAS+ representation for the belief is built as the merge of each state's representation. Finally, `FD` is called with this file.

We see several benefits to this approach. First, it is very fast, as the translation of each problem P_i is very simple. Comparatively, the translation of $P_{1...i}$ can be more difficult because it is not always easy to determine, e.g., how to partition the facts in order to compute the best SAS+ variables. Second, it allows us to avoid the use of the `forall` PDDL construct which `FD` seems to have a hard time dealing with (we understand that the `forall` could be explicitly unfolded, and all the cases that it represents be enumerated, but this solution lacks elegance). Third, because each P_i problem is very small, we could actually spend extra resources to optimise the SAS+ encoding; this is something that would be riskier to do for the full problem $P_{1...i}$.

[2]This is useful in particular because some variants of `CPCES` remove and re-add counter-examples from the sample set (Grastien and Scala 2018).

Merging Variables

Inspired from merging facts in PDDL, merging variables in SAS+ works as well. Thanks to certain-facts, we are able to find those variables that contain all the facts are initially known and have no influence on uncertain-facts. Certain-facts have a fixed Boolean value from initial state to goal state. If all the facts in a variable are certain-facts, we just keep this variable once. By merging variables, FD can reduce the search space and greatly improve the search efficiency when searching for a plan.

Experimental Results

The benchmarks and the experimental setup are the same as Section 4. The search engine used by `FD` is eager best-first search algorithm, best-first open list with FF heuristic. Table 2 shows the results. We compare column NMF that represents the merged version (Section 4) of `CPCES` using `FF`, which we refer to as `CPCES(FF)` here; column FD that represents the basic `CPCES` with `FD`, referred to as `CPCES(FD)`; and column ICC that represents the incremental version `CPCES`, referred to as `I-CPCES(FD)`. We also refer to Figure 6 for a comparison between the length of the plans computed by `CPCES(FF)` and `I-CPCES(FD)`.

Comparing `CPCES(FD)` and `I-CPCES(FD)`, we see a clear improvement in domains such as BOMB, COINS, DISPOSE, and ONEDISPOSE. Other domains such as LOOK-GRAB have more noisy results for which it is hard to draw definite conclusions.

`I-CPCES(FD)` is competitive against `CPCES(FF)`. `I-CPCES(FD)` solves DISPOSE and ONEDISPOSE faster than `CPCES(FF)`. In addition, `I-CPCES(FD)` finds shorter paths (e.g., 419 against 633 for DISPOSE p-8-3 or 66 against 139 for ONEDISPOSE p-3-3).

`I-CPCES(FD)` struggles with more complex domains such as BLOCKWORLD, BOMB, LOOK-GRAB, and RAOSKEYS. However, we note that `I-CPCES(FD)` finds a much shorter plan than `CPCES(FF)` for the instance p-8-1-1 of LOOK-GRAB (126 against 174). This leads us to believe that `FD` may be trying too hard to compute a short plan. Figure 6 also shows that the plan length from `FD` is normally less than the length from `FF`. This phenomenon can explain why the total performance of searching time of `I-CPCES(FD)` is worse than `CPCES(FF)`, as is shown on Figure 7.

7 Conclusion

In this article, we introduce three improvements to `CPCES`. First, we identify certain-facts, PDDL facts whose value is never uncertain throughout an execution. We merge these facts in the reduction to classical planning used during the candidate plan generation. Second, we propose a warm-start

DM	Instance	Total Time (s)					Iterations					Plan Length		RF
		NMF	MF	WS	FD	ICC	NMF	MF	WS	FD	ICC	NMF	ICC	(%)
BW	01	0.04	0.04	0.05	0.12	0.08	3	3	3	3	3	5	4	0
BW	02	0.18	0.18	0.15	0.48	0.29	7	8	6	8	8	20	15	0
BW	03	1.70	1.78	1.40	-	-	27	30	23	-	-	57	-	0
BW	04	146.72	231.65	133.07	-	-	81	86	77	-	-	152	-	0
BB	20-10	0.72	0.72	0.20	11.61	1.16	20	20	3	20	20	37	37	0
BB	20-20	1.08	1.05	0.36	24.66	1.99	20	20	4	20	20	37	37	0
BB	100-1	33.36	22.84	1.05	-	21.27	100	100	2	-	100	197	197	0
BB	100-5	69.12	48.12	3.45	-	341.24	100	100	3	-	100	197	197	0
BB	100-10	115.49	71.89	5.14	-	897.62	100	100	3	-	100	197	197	0
BB	100-60	792.81	412.91	27.60	-	-	100	100	3	-	-	197	-	0
BB	100-100	1660.01	872.93	90.08	-	-	100	100	4	-	-	197	-	0
CN	15	0.91	0.87	0.90	5.76	1.07	15	15	15	21	17	80	78	0
CN	16	0.75	0.78	0.72	3.00	0.91	10	10	9	11	10	109	114	0
CN	17	0.65	0.68	0.73	2.87	0.97	9	9	9	11	11	100	111	0
CN	18	0.63	0.63	0.69	3.21	0.90	9	9	9	12	11	103	97	0
CN	19	0.81	0.79	0.81	2.88	0.89	10	10	10	11	10	110	107	0
CN	20	0.77	0.78	0.84	2.90	0.89	9	10	9	10	9	111	109	0
DP	4-2	1.05	0.95	0.14	3.59	1.16	17	17	2	17	17	74	72	33
DP	4-3	1.29	1.28	0.18	4.11	1.46	17	17	2	17	17	88	89	25
DP	8-1	37.19	16.02	1.13	723.66	15.73	65	65	2	65	65	347	259	50
DP	8-2	130.98	82.56	2.37	901.25	57.32	65	65	2	65	65	494	354	33
DP	8-3	498.74	203.84	10.13	1243.76	86.34	65	65	2	65	65	633	419	25
DP	12-1	-	508.48	11.06	-	512.04	-	145	2	-	145	-	697	50
DP	12-2	-	-	124.35	-	1670.40	-	-	2	-	145	-	912	33
DP	16-1	-	-	133.99	-	-	-	-	2	-	-	-	-	50
LG	4-3-3	0.18	0.18	0.36	65.29	83.65	2	2	2	2	2	4	4	20
LG	8-1-1	8.01	5.03	4.13	415.49	168.66	20	18	2	22	26	174	126	33
LG	8-1-2	2.23	1.57	3.27	-	-	12	12	2	-	-	104	-	33
LG	8-1-3	1.61	1.25	2.54	-	-	11	11	2	-	-	46	-	33
LG	8-2-1	29.38	12.67	11.70	543.13	-	22	19	2	19	-	104	-	25
LG	8-2-2	4.85	3.27	11.34	-	-	11	12	2	-	-	94	-	25
LG	8-3-1	30.05	15.65	25.76	460.49	-	16	15	2	15	-	106	-	20
LG	8-3-2	7.63	5.04	22.35	-	-	12	11	2	-	-	60	-	20
OD	2-3	0.50	0.33	0.06	0.92	0.48	17	14	2	13	11	47	27	20
OD	3-2	2.50	1.31	0.09	4.09	1.15	29	26	2	28	26	74	44	25
OD	3-3	47.36	234.88	0.29	17.06	3.02	46	56	4	59	51	139	66	20
OD	4-2	-	117.27	1.32	73.25	4.39	-	65	11	80	56	-	80	25
OD	4-3	-	-	-	-	657.89	-	-	-	-	101	-	115	20
OD	5-2	1491.01	401.45	171.42	-	43.00	152	141	101	-	140	269	146	25
RK	02	0.07	0.07	0.06	0.24	0.14	5	5	4	5	5	16	17	14
RK	03	0.72	0.59	0.57	61.04	13.07	21	20	19	37	37	48	66	11

Table 2: Performance of the different algorithms presented in this paper on selected instances. This table presents the runtime, number of iterations, and plan length, as well as the ratio of certain-facts in the instance. NMP is the classic CPCES; MP is the variant with merging facts; WS uses warm-starting; FD is CPCES with FD; ICC is incremental CPCES with FD. In the column of domain (DM), BW is BLOCKWORLD, BB is BOMB, CN is COINS, DP is DISPOSE, LG is LOOK-GRAB, OD is ONEDISPOSE, RK is RAOSKEYS. Last column shows the ratio of certain-facts (RF) of each problem.

version of CPCES in which some sample states are already included in the sample set. Finally, we discuss the use of FD in CPCES as FD has traditionally been performing poorly compared to FF. We assumed that the problem lies, at least to some degree, with the generation of the SAS+ representation of the PDDL problem, and proposed an incremental generation which not only is faster but also leads to better representations. Our experiments, summarised in Figure 7, validate these changes and show improved performance.

This work opens new research perspectives. For instance, we currently use an approximate method based on contexts to identify certain-facts. We could develop more methods that are not purely based on the structure of the problem. We also managed to make FD competitive as the classical planner that powers CPCES. This is encouraging for two reasons: First, because FD gives us access to a large range of planning options. While we were unable to find a setup better than FF, there is much to be done here. Second, this is a first successful attempt at exploiting the peculiar structure of the classical planning problems generated by CPCES. We believe that further improvements could be identified.

References

Albore, A.; Palacios, H.; and Geffner, H. 2010. Compiling uncertainty away in non-deterministic conformant planning. In *Nineteenth European Conference on Artificial Intelligence (ECAI-10)*, volume 215, 465–470.

Albore, A.; Ramirez, M.; and Geffner, H. 2011. Effective heuristics and belief tracking for planning with incomplete information. In *21st International Conference on Automated Planning and Scheduling (ICAPS-11)*.

Bäckström, C.; and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4): 625–655.

Bonet, B.; and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Fifth International Conference on AI Planning Systems (AIPS-00)*, 52–61.

Cimatti, A.; and Roveri, M. 2000. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13: 305–338.

De Moura, L.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Fourteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-08)*, 337–340. Springer.

Geffner, H. 2007. The causal graph heuristic is the additive heuristic plus context. In *ICAPS Workshop on Heuristics for Domain-Independent Planning*.

Grastien, A.; and Scala, E. 2017. Intelligent belief state sampling for conformant planning. In *26th International Joint Conference on Artificial Intelligence (IJCAI-17)*, 4317–4323.

Grastien, A.; and Scala, E. 2018. Sampling strategies for conformant planning. In *28th International Conference on Automated Planning and Scheduling (ICAPS-18)*.

Grastien, A.; and Scala, E. 2020. CPCES: A planning framework to solve conformant planning problems through a counterexample guided refinement. *Artificial Intelligence*, 284: 103271.

Haslum, P.; and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *Fifth European Conference on Planning (ECP-99)*, 308–318.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6): 503–535.

Hoffmann, J.; and Brafman, R. I. 2006. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7): 507–541.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Kurien, J.; Nayak, P. P.; and Smith, D. E. 2002. Fragment-based conformant planning. In *Sixth International Conference on AI Planning Systems (AIPS-02)*, 153–162.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL: The Planning Domain Definition Language. *Technical Report*.

Nguyen, K. H.; Tran, V. D.; Son, T. C.; and Pontelli, E. 2012. On computing conformant plans using classical planners: A generate-and-complete approach. In *22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*.

Palacios, H.; and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35: 623–675.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *23rd Conference on Artificial Intelligence (AAAI-08)*, volume 8, 975–982.

Richter, S.; and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. In *International Planning Competition (IPC-11)*, 50–54.

Rintanen, J. 2012. Engineering efficient planners with SAT. In *20th European Conference on Artificial Intelligence (ECAI-12)*, 684–689.

Smith, D. E.; and Weld, D. S. 1998. Conformant graphplan. In *Fifteenth Conference on Artificial Intelligence (AAAI-98)*, 889–896.

To, S. T.; Pontelli, E.; and Son, T. C. 2011. On the effectiveness of CNF and DNF representations in contingent planning. In *22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*.

Zhang, X.; Grastien, A.; and Scala, E. 2020. Computing superior counter-examples for conformant planning. In *34th Conference on Artificial Intelligence (AAAI-20)*, 10017–10024.