Sampling Strategies for Conformant Planning

Alban Grastien,[1,2] **Enrico Scala**[2,3]
[1] Data61, Canberra, Australia
[2] The Australian National University, Canberra, Australia
[3] Fondazione Bruno Kessler, Trento, Italy

Abstract

We present a generalisation of CPCES, a conformant planner that uses two procedures: candidate plan generation and sampling of the initial belief state. The new CPCES better distinguishes these two procedures and therefore provides a clearer framework for the resolution of conformant planning problems. We study CPCES theoretically by analysing the sampling phase through the lens of *tags*, *width* and *basis*. The benefit of this new interpretation is twofold: firstly it allows us to bound the maximum number of iterations required by CPCES, and second it allows us to individuate sampling strategies that guarantee the discovery of subsets of *minimal* bases. An experimental analysis reported in the paper shows that the greedy sampling (the original version of CPCES) is the more effective strategy, coverage wise. However, when either the quality of the plans or the size of the resulting samples is important a more sophisticated sampling is more effective.

1 Introduction

Conformant planning is the problem of finding a valid plan in an uncertain environment (Smith and Weld 1998). We consider here deterministic conformant planning, i.e., the variant where uncertainty is only encountered in the initial state, and not in the effects of the actions.

Consider the problems of Fig. 1 where an agent with unknown initial location must reach the center (g) position. The agent may move in all four directions north (N), south (S), east (E), and west (W), and stays in its current position if it hits a wall. The example on the right also includes a swamp and a wall right next to it (it is impossible to leave the swamp, but the agent is initially outside it). A solution for instance a. is $(\text{E}\times4)(\text{W}\times2)(\text{N}\times4)(\text{S}\times2)$ and for instance b. is $(\text{E}\times4)(\text{N}\times4)(\text{E}\times2)(\text{W}\times2)(\text{S}\times2)$. Interestingly, in instance a. the solutions are precisely the plans valid for two opposite corners (e.g., $\langle 0,0\rangle$ and $\langle 4,4\rangle$); these two initial states form a (minimal) *basis* (Albore, Ramírez, and Geffner 2011). This implies for instance that some of the uncertainty about the initial situation (i.e., the agent could be in one of the locations not surrounded by any walls) can be ignored.

Leveraging the idea that, actually, many of those possible initial states are not relevant, the recently proposed conformant planner CPCES (Grastien and Scala 2017a) uses a counter-example based refinement strategy that works as follows. CPCES firstly computes a *candidate plan* valid for a *sample* of the initial belief state; then it tries to validate this plan through a process that generates a *counter-example*, i.e., an initial state for which the plan is invalid; this counter-example is then added to the sample (which is initially empty) and a new candidate plan is generated until a valid plan is produced or it is proved that there is no plan.

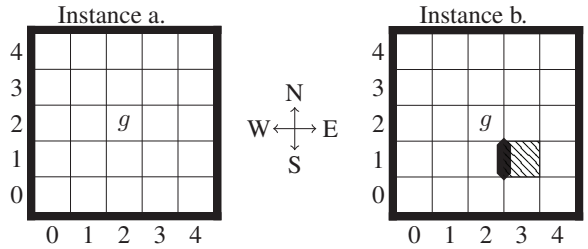

Figure 1: Two conformant planning problems: find a conformant path to location g; thick black lines are walls; striped regions are swamps.

Complementary to other attempts done in solving conformant planning problems (e.g, (Cimatti, Roveri, and Bertoli 2004; To, Pontelli, and Son 2009; Castellini, Giunchiglia, and Tacchella 2003; Palacios and Geffner 2009)), we provide an in-depth theoretical investigation of CPCES. We study the connection between this approach and known concepts from the conformant planning literature: *bases*, *tags*, and *width* (Bonet and Geffner 2014; Albore, Ramírez, and Geffner 2011). Tags are statements about the initial state that are sufficient to track the value of a variable during the execution: for instance in instance a. the initial horizontal position of the agent is sufficient to determine its horizontal position after any sequence of actions; this no longer applies to instance b. as the vertical component also matters to determine if the swamp or a wall was reached. Finally the width represents the maximum number of variables that appear in some tag: it is one for Fig. 1.a and two for Fig. 1.b.

Our theoretical analysis unearthes a number of interesting insights. First, we present a generalisation of CPCES that provides flexibility in the way the sample is computed; it allows us to try to identify states that can be ignored. Second, we characterise the output of the sample of CPCES with re-

spect to the notion of basis. Third, we provide complexity results for CPCES, given in terms of the width and the number of tags. In particular, we show that *dominance-preserving* sampling leads CPCES to converge, in the worst case, in a number of steps that is linear with the number of tags, and so exponential in the width of the problem. Finally, we provide variants of CPCES that propose different strategies to compute samples. One of such allows us to only generate *minimal* bases subsets. This helps in explaining unsolvable problems, and limiting the burden of our plan generation phase. The next section reports formal definition of conformant planning and notions relevant to this work. Section 3 describes the generalised version of CPCES and individuates different classes of sampling. Section 4 studies the previous algorithms theoretically, while the last section studies them from an empirical standpoint.

2 Preliminaries

In this section we present the problem of conformant planning and the CPCES algorithm of Grastien & Scala (2017a).

2.1 Conformant Planning Problem

Following Geffner and Bonet (2014), a *(deterministic) conformant planning problem* is a tuple $P = \langle \mathcal{V}, A, \Phi_I, \Phi_G \rangle$ where $\mathcal{V}$ is a set of *state variables* v each with a domain D_v, A is a set of *actions*, Φ_I is the *initial belief*, and Φ_G is the *goal formula*. A *variable assignment* is a variable v and one of its possible values: $d \in D_v$. An action $a \in A$ is a pair $\langle p, \mathit{eff} \rangle$ where p is a precondition on the values of $\mathcal{V}$ and eff is a function (called the effects of a) that associates each possible assignment of each variable with a precondition on the values of $\mathcal{V}$. We assume that the conditions of two inconsistent effects are inconsistent: $\forall v \in \mathcal{V}.\, \forall \{d_1, d_2\} \subseteq D_v.\, d_1 \neq d_2 \Rightarrow \mathit{eff}(v, d_1) \wedge \mathit{eff}(v, d_2) \models \bot$. We further assume that Φ_G and the precondition of each action are conjunctions of variable assignments

A *state* is a complete assignment of the state variables. An action $a = \langle p, \mathit{eff} \rangle$ is *applicable* in state s if its precondition is satisfied: $s \models p$. The result of *applying* a in s is the state $s[a]$ in which the variable v is assigned to d iff either $\mathit{eff}(v, d)$ holds in the current state s, or v was already assigned to d in s and $\mathit{eff}(v, d')$ holds in s for no $d' \in D_v$.

A *belief state* Φ is a Boolean formula on the value of $\mathcal{V}$ (Φ_I is such a belief state) that represents the set of states consistent with Φ. We use indiscriminately the formula or its set of states to represent a belief state; in the latter case we use the symbol $\mathcal{B}$.

Action $a = \langle p, \mathit{eff} \rangle$ is *applicable* in belief state Φ iff it is applicable in every one of its states or, equivalently, if $\Phi \models p$ holds. The result of *applying* action a in the belief state Φ is the set of states defined by applying a in every state of Φ: $\Phi[a] = \{s[a] \mid s \in \Phi\}$.

A *plan* $\pi = a_1 \dots a_k$ is a (possibly empty) sequence of actions. The *application* $\Phi[\pi]$ of the plan is the incremental application: $\Phi[a_1] \dots [a_k]$. The plan is *applicable* if each of its actions a_i is applicable in $\Phi[a_1] \dots [a_{i-1}]$.

A *solution* to the planning problem is a plan π that is applicable from Φ_I and that leads to the goal set: $\Phi_I[\pi] \models \Phi_G$; we say that a solution is a *valid* plan. We write $\Pi(P)$ the set of solutions to the planning problem P. A solution is *optimal* if there is no shorter solution. Given a belief state Φ, we write P_Φ the planning problem $\langle \mathcal{V}, A, \Phi, \Phi_G \rangle$ (i.e., where Φ_I is replaced with Φ).

A planning problem is *classical* if it has a single initial state. Classical problems are generally easier to solve than conformant ones (PSPACE-complete (Bylander 1991) vs EXPSPACE-hard (Haslum and Jonsson 1999; Rintanen 2004)).

2.2 Relevant Concepts of Conformant Planning

A *basis* (Palacios and Geffner 2009) is a subset $\mathcal{B}$ of initial states such that $\Pi(P_\mathcal{B}) = \Pi(P)$. We say that a basis is *minimal* if none of its proper subset is a basis.

We also introduce the notion of optimality-basis. An *optimality-basis* is a subset $\mathcal{B}$ of initial states such that the optimal plan for $\mathcal{B}$ is the optimal plan for the conformant planning problem: $\arg\min \Pi(P_\mathcal{B}) = \arg\min \Pi(P)$. Notice that all bases are optimality-bases too, although a minimal basis may not be a minimal optimality-basis.

Bases are interesting objects, in particular because they provide a "justification" for the selected plan. Indeed for each invalid plan that a user might come up with, there is a state in the basis for which this plan is invalid. Optimality-bases are also useful, because they justify the optimality of the solution: for any invalid plan that is cheaper than the optimal solution, there is a state in the optimality-basis for which this plan is invalid. If the (optimality-)basis is small this verification can be made manually (i.e., by a human). The size of the basis is therefore crucial: for instance Φ_I is a basis, but hardly a useful one. For this reason we are interested in subset-minimal bases.

The width of a conformant planning problem is a measure of its complexity (Bonet and Geffner 2014). Intuitively, the width captures interactions among uncertain variables. The *context* $ctx(v)$ of a variable is the set of *relevant variables* of v defined iteratively by:

- v is relevant to v;
- if $\mathit{eff}(v_1, d)$ is different from $\bot$ for some action $\langle p, \mathit{eff} \rangle \in A$ and v_2 appears in $\mathit{eff}(v_1, d)$ or in the precondition p, then v_2 is relevant to v_1;
- if v_2 is relevant to v_1 and v_3 is relevant to v_2, then v_3 is relevant to v_1 (relevance is transitive).

The *width* of a variable v is the number of elements in $ctx(v) \cap \mathcal{V}_u$ where $\mathcal{V}_u$ is the set of variables that are initially uncertain. The width w of a conformant planning problem is the largest width of any of the variables in its goal or action preconditions.

There are two important results linking the width to the complexity of conformant planning. Firstly there exists a basis of size exponential in the width (Bonet and Geffner 2014). The second known result (Palacios and Geffner 2007) is a correct and complete reduction (called *KTM*) of conformant planning to classical planning that multiplies the number of state variables by a factor exponential in w. KTM relies on the notion of tags and merges that we describe now.

For a variable v and a belief state $\mathcal{B}$ a *tag* $t \in \mathit{Tags}(\mathcal{B}, v)$ is a complete assignment of the state variables in $ctx(v)$ that

can be true in $\mathcal{B}$. The number of tags is exponential in w and linear in v. A *merge* is the set of tags generated for each variable ($Tags(\Phi_I, v)$ in our notation): in other words in each merge, exactly one tag is initially true. For each assignment $\langle v, d\rangle$ and each tag t KTM introduces the propositions $K(v = d)_{/t}$ and $K\neg(v = d)_{/t}$ that can be interpreted as follows: if t was true in the initial state, then v is known to now evaluate (or not) to d. We refer to (Palacios and Geffner 2007) for the complete details on the reformulation.

2.3 CPCES

CPCES is a conformant planner presented by Grastien & Scala (2017a). It relies on the following property: $\Pi(P_{\mathcal{B}_1 \cup \mathcal{B}_2}) = \Pi(P_{\mathcal{B}_1}) \cap \Pi(P_{\mathcal{B}_2})$. In particular $\mathcal{B} \models \Phi_I$ implies $\Pi(P_{\mathcal{B}}) \supseteq \Pi(P)$, which can be interpreted as follows: a plan invalid for a subset of initial states is invalid for the complete problem. CPCES interweaves two procedures.

The first procedure generates a *candidate plan* for a subset of the initial states (i.e., $\mathcal{B} \models \Phi_I$); subset of initial states called a *sample*. In practice this procedure is solved by reducing conformant planning (with a small number of initial states) to classical planning. Initially the sample is empty ($\mathcal{B} = \emptyset$) and the first plan is generally empty ($\pi = \varepsilon$).

The second procedure computes a *counter-example*, i.e., a state $q \in \Phi_I$ for which the candidate plan is invalid. This problem is CO-NP-complete (Grastien and Scala 2017b) so in practice it is solved by using a SAT solver.

The sample is defined as the set of all counter-examples computed so far, so that the candidate plan is guaranteed to be different at each iteration. When one of the two procedures fails to find a solution either the problem has been proved unsolvable (failure to generate a candidate plan) or it has been solved (failure to find a counter-example to the—hence valid—candidate plan).

CPCES has been proved sound and complete, and scales much better than the other standard algorithms for problems with a non-trivial width.

Consider the example of Fig. 1.a. Starting with an empty plan ε, CPCES *might produce the counter-example $\langle 0, 0\rangle$: the empty plan does not lead the agent to g if the agent starts in location $\langle 0, 0\rangle$. The plan generator might then propose plan* EENN*; the initial state $\langle 1, 0\rangle$ is however a counter-example to this plan. A plan that would apply to both counter-examples is then* WEENN*. More counter-examples can be produced, but we assume here that the state $\langle 4, 4\rangle$ is now exhibited. The next candidate plan is* (E×4)(W×2)(N×4)(S×2)*. Since this plan is valid, no counter-example is produced and the solution is returned.*

3 CPCES: A Family of Conformant Planner

In this section we present the CPCES family as summarised in Fig. 2. Arrows in the figure indicate implementations: for instance, dpCPCES is an implementation of the general CPCES. Properties enjoyed by the different variants are discussed in Section 4.

3.1 Generalised CPCES

A result of our study of CPCES is that it is an instance of a more general class of algorithms that we present now.

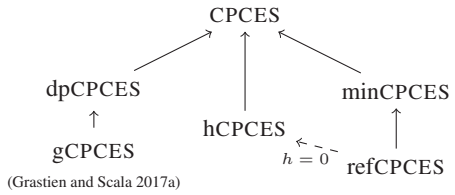

Figure 2: The CPCES Family. The optimal variant is orthogonal and therefore not represented on the graph.

We use the notation $\mho = \{\pi_1, \ldots, \pi_m\}$ to refer to a set of (invalid) plans. Let $\mathcal{B} = \{q_1, \ldots, q_k\}$ be a subset of initial states. We say that $\mathcal{B}$ is a *counter-example set* of $\mho$ if $\mathcal{B}$ contains at least one counter-example for all plans in $\mho$: $\forall i \in \{1, \ldots, m\}.\ \exists j \in \{1, \ldots, k\}.\ \pi_i \notin \Pi(P_{\{q_j\}})$. Equivalently: $\mho \cap \Pi(P_{\mathcal{B}}) = \emptyset$. A counter-example set is *minimal* for $\mho$ if no strict subset is a counter-example set of $\mho$.

Consider again the execution of CPCES *for Fig. 1.a as presented before. The belief state $\mathcal{B}_1 = \{\langle 0, 0\rangle, \langle 1, 0\rangle\}$ is a counter-example set for the set of invalid plans $\mho = \{\varepsilon,$* EENN*$\}$. However, so is $\mathcal{B}_2 = \{\langle 1, 0\rangle\} \subset \mathcal{B}_1$. Therefore $\mathcal{B}_1$ is not minimal for $\mho$.*

Based on this notion of counter-example set, we define a new version of CPCES (Algo. 1). Given a current sample $\mathcal{B}$ CPCES calls the procedure `candidate-plan` that generates a candidate plan π. This plan is added to $\mho$ and a new counter-example set $\mathcal{B}$ is calculated; if no such $\mathcal{B}$ exists, then the new candidate plan is valid and returned by CPCES.

Algorithm 1 A new CPCES.

```
1: ℧ := ∅
2: B := ∅
3: loop
4:    π := candidate-plan(B)
5:    if there is no such π then
6:       return there is no plan
7:    ℧ := ℧ ∪ {π}
8:    B := counter-example-set(℧)
9:    if there is no such B then
10:      return π
```

CPCES as defined by Grastien & Scala (2017a) is an instantiation of Algo. 1. Indeed this algorithm implements Line 8 by collecting counter-examples in a greedy fashion:

```
8.1:  q := counter-example(π)
8.2:  B := B ∪ {q}
```

For this reason, we refer to the original algorithm from Grastien & Scala as gCPCES, i.e., greedy CPCES.

If the candidate generator (Line 4) returns the optimal plan for the current sample, we prefix CPCES with o-.

3.2 Dominance-Preserving CPCES

In CPCES the function of the sample is to reduce the set of valid plans so that the plan generated on Line 4 is valid

for the conformant problem. A relevant property is therefore the fact that the set of valid plans for the sample decreases monotonically at each iteration of CPCES.

We say that belief state $\mathcal{B}$ *dominates* belief state $\mathcal{B}'$ if its valid plans form a subset of those of $\mathcal{B}'$: $\Pi(P_{\mathcal{B}}) \subseteq \Pi(P_{\mathcal{B}'})$.

A sampling strategy is said to be *dominance-preserving* if at each iteration it keeps a counter-example set that dominates all the previous samples (or equivalently, that dominates the previous sample). We write dpCPCES the class of implementations of CPCES that use a dominance-preserving strategy.

gCPCES is a dominance-preserving dpCPCES. Indeed in gCPCES the sample is a super set of the sample at the previous iteration; its set of solutions is therefore a subset of the set of solutions at the previous iteration.

3.3 Minimised CPCES

We mentioned that the role of the sample is to reduce the number of valid solutions. A different ambition is to reduce the size of the sample, either for computational purposes (the candidate plan generation is simpler if the size of the sample is smaller) or to provide a smaller justification.

We say that a state $q \in \mathcal{B}$ is *redundant* for $\mho$ if all plans of $\mho$ that are invalid for q are also invalid for some other state of $\mathcal{B}$. A counter-example set $\mathcal{B}$ is minimal iff none of its states is redundant, since one such state could be removed from $\mathcal{B}$ without affecting its coverage.

We call minCPCES an implementation of CPCES that computes a minimal counter-example set in Line 8. We present refCPCES, an implementation of minCPCES. As gCPCES, refCPCES only differs from CPCES by instantiating its Line 8:

8.1: $q := \texttt{counter-example}(\pi)$
8.2: $\mathcal{B} := \mathcal{B} \cup \{q\}$
8.3: $\mathcal{B} := \texttt{minimise}(\mathcal{B}, \mho)$

The method of Line 8.3 computes a subset of the current sample $\mathcal{B}$ that forms a minimal counter-example set of $\mho$. This procedure iteratively verifies whether the sample includes a redundant state. The minimisation procedure is presented in Algorithm 2. Notice that the procedure requires to check whether a given plan is valid from a given state, which is very simple to verify.

refCPCES guarantees that the sample is always minimal; it is therefore an implementation of minCPCES. Compared to a dpCPCES implementation, refCPCES proposes a very aggressive strategy to reduce the sample. This strategy can be counter-productive as the counter-examples that are discarded may be relevant.

We consider (a simplified variant of) the `dispose` *domain, and the instance where the single item is in one of n locations and must be picked up before being disposed of; the initial location of the item is unknown. The optimal plan π^* consists in visiting all locations to pick up the item before disposing of it. For any subset of initial states, a plan shorter than π^* exists: skip the pick-up action in the locations that do not contain the item. Therefore an implementation of o-* CPCES *terminates only when π^* is generates, i.e., when the sample is exactly the set of initial states.*

Algorithm 2 Minimising the sample.

```
 1: procedure minimise_sample(B)
 2: for all q ∈ B do
 3:    if state_is_redundant(q,B \ {q}) then
 4:       B := B \ {q}
 5: return B
 6:
 7: procedure state_is_redundant(q,Q)
 8: for all π ∈ ℧ do
 9:    if π is invalid for q then
10:       if ¬ has_counter_example(π, Q) then
11:          return false {q is the only counter-example to π}
12: return true
13:
14: procedure has_counter_example(π,Q)
15: for all q' ∈ Q do
16:    if π is invalid for q' then
17:       return true
18: return false
```

Refined sample	Plan	Counter-example
$\emptyset$	ε	$L1$
$\{L1\}$	G1 P D	$L2$
$\{L2\}$	G2 P D	$L1$
$\{L1, L2\}$	G1 P G2 P D	$L3$
$\{L3\}$	G3 P D	$L1$
$\{L1, L3\}$	etc.	

Table 1: Execution of o-refCPCES for `dispose`: Li is the initial state where the item is in the ith location, G stands for goto, P for pick up, and D for dispose.

*Let us look at the execution of o-g*CPCES*. From the initial plan (ε) is produced the counter-example $L1$ in which the item is in the first location. A plan is generated for this counter-example (go to location 1, pick-up item, dispose of item). A counter-example $L2$ is produced in which the item is in the second location. A more sophisticated plan is now generated (go to location 1, pick-up item, go to location 2, pick-up item, dispose of item). This procedure goes on until all possible initial locations have been accounted for.*

*Now let us look at the execution of o-ref*CPCES*. This execution is summarised in Table 1; notice that the refined sample at any time step is a minimal subset of the sample at the previous time step plus the last counter-example that contradicts all plans found so far. The execution is originally similar, but then the algorithm notices that $L2$ is a counter-example of the first plan (ε). Therefore the algorithm concludes that $L1$ is redundant. A new plan is produced for $\{L2\}$, which leads to the counter-example $L1$ being produced again. This time, $L1$ is not redundant and a plan must be produced for $\{L1, L2\}$. The counter-example $\{L3\}$ is then produced, which allows the algorithm to ignore the previous two counter-examples. It is possible to prove that all subsets of initial states will eventually be generated as samples.*

Algorithm 3 Heuristic Refinement.

```
1: for all q ∈ B do
2:   if state_is_redundant(q,B \ {q}) then
3:     if h+(B \ {q}) = h+(B) then
4:       B := B \ {q}
```

3.4 hCPCES

We argued in the previous subsections (and prove formally in the next section) that dominance-preserving strategies require fewer iterations than worst-case CPCES. We also argued that reducing the size of the sample is beneficial. It is possible to define a strategy that minimises the sample while preserving dominance. Computing such sample is however hard[1] so we settle for a heuristic approach instead.

We say that a counter-example q is *useful* for $\mathcal{B}$ iff $\mathcal{B} \cup \{q\}$ restricts the set of solutions, i.e., the resulting set dominates the starting one: $\Pi(P_{\mathcal{B}\cup\{q\}}) \subset \Pi(P_{\mathcal{B}})$. We approximate this reachability problem by using a delete-free relaxation based heuristic of the classical reduction of Π given a counter-example set $\mathcal{B}$; we call such a relaxation $\Pi^+(P_{\mathcal{B}})$, and its optimal solution $h^+(P_{\mathcal{B}})$.[2] We say that q is *probably-useful* for $\mathcal{B}$ if $h^+(P_{\mathcal{B}})$ is lower than $h^+(P_{\mathcal{B}\cup\{q\}})$. This gives us guarantees that the addition of q is at least contradicting all the plans of length i) strictly less than $h^+(P_{\mathcal{B}\cup\{q\}})$ and ii) longer than $h^+(P_{\mathcal{B}})$. These plans were less obviously contradicted before. The idea is to use this heuristic notion of usefulness to reconsider a reasonable subset of counter-examples discarded by refCPCES. Algo. 3 sketches its overall functioning.

Nicely, refCPCES can be seen as a special case of this approach: it suffices to substitute the h^+ function with a trivial admissible heuristic returning 0 for each input.

We name the resulting sampling strategy hCPCES.

On the negative side however, hCPCES only approximates a dominance preserving sampling strategy: There could be counter-examples that the heuristic does not consider as helpful, but that in principle would lead to the actual dominant counter-example set for that iteration.

Finally, for alleviating the computational burden within the sampling, we approximate h^+ using the h_{FF} heuristic[3].

We illustrate hCPCES *with* `dispose`*. Assume we are given the counter-examples $L1$ and $L2$ presented before. In ref*CPCES*, $L1$ would be removed from the sample because it is redundant. Consider the delete-free relaxation of the conformant problem with initial belief $\{L2\}$; the relaxed plan of $\{L2\}$ involves going to the second location, picking up the item from the second location, and disposing of it. If the initial belief is $\{L1, L2\}$, the relaxed plan also involves navigating to the first location (which could be in the way of the second location, so it might not increase the length of the relaxed plan) and picking up the item from the first location (the PDDL action is distinct from the one in the second location).* hCPCES *therefore keeps $L1$ in the sample.*

We now consider the example of Fig. 1.a. Assume that the current sample is $\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 4,4\rangle\}$. Clearly the middle state is not useful. The delete-free relaxation will compute plan EENNWWSS *which, while incorrect, is the same as the delete-free relaxed plan for $\{\langle 0,0\rangle, \langle 4,4\rangle\}$.* hCPCES *therefore correctly ignores the middle state.*

[1]Proving $\Pi(P_{\mathcal{B}\setminus\{q\}}) = \Pi(P_{\mathcal{B}})$, i.e., determining if q can be removed from $\mathcal{B}$ while preserving dominance, is PSPACE-complete for an enumerated set $\mathcal{B}$; proof omitted here.

[2]This heuristic is computed by first translating the problem into a classical planning problem following (Grastien and Scala 2017a); then applying the delete-free relaxation (Bonet and Geffner 2001).

[3]Let us recall that solving the delete-free relaxation problem optimally is NP-Hard (Bylander 1991).

4 Properties of CPCES

4.1 Termination

The proof of termination of gCPCES (Grastien and Scala 2017a) relied on the fact that an initial state is added to the sample at each iteration, together with the fact that there are finitely many initial states. This fact no longer holds in CPCES. Yet, we can still retain soundness and completeness as long as plan generation is correct and the sampling correctly generates a counter-examples set.

Theorem 1 *If the procedures* `candidate-plan` *and* `counter-example-set` *are correct,* CPCES *(Algo. 1) always terminates, and is sound and complete.*

Proof: If the algorithm terminates, then clearly the result is correct (because either it returns a plan for which there is no counter-example or it notifies that no plan was found for a sample of the initial states).

Secondly, consider the sample $\mathcal{B}_i \subseteq states(\Phi_I)$ at some iteration of CPCES; consider the candidate plan π computed for $\mathcal{B}_i$; and assume that the plan is not valid for Φ_I. Then π will be added to $\mho$. Consequently, at any later iteration, the sample $\mathcal{B}_j$ (a counter-example set for $\mho \supseteq \{\pi\}$) will be different from $\mathcal{B}_i$. Since there is a finite number ($2^{|states(\Phi_I)|}$) of possible samples the algorithm eventually terminates. □

Remark that Th. 1 carries over to all implementations of CPCES.

4.2 Number of Iterations

From the proof of Th. 1 we can deduce that the maximum number of iterations is $2^{|\Phi_I|}$. We present better bounds however.

Given a belief state $\mathcal{B}$, a variable v and a tag $t \in Tags(\mathcal{B}, v)$ (remember that t assigns a value to all variables in $ctx(v)$) the *projection* of the planning problem P on t is the (classical) planning problem $P_{v,t} = \langle ctx(v), A', t, \Phi'_G\rangle$ defined as follows:

- For each action $a = \langle p, \mathit{eff}\rangle \in A$ there is an action $a' = \langle p', \mathit{eff}'\rangle \in A'$ where p' is the subset of variables assignments of p that involve $ctx(v)$ and $\mathit{eff}'(v', d) = \mathit{eff}(v', d)$ for all variable $v' \in ctx(v)$ and $d \in D_v$.
- Φ'_G is the restriction of Φ_G to the variables of $ctx(v)$.

We assume that the actions are still the same ($A' = A$) although their semantics are modified, so that the set of valid plans of $P_{v,t}$ can be compared to that of P.

Using the example of Fig. 1.a, consider the variable x that represents the horizontal position of the agent. Remember: $ctx(x) = \{x\}$. Consider the tag $t = (x = 1)$ (the agent is

in column 1*). In the problem* $P_{x,t}$ *the actions* W *and* E *are unchanged while* N *and* E *become blank actions. The goal of* $P_{x,t}$ *is to reach* $x = 2$ *(the vertical position is ignored in the goal). The set of plans for this problem includes:* E, WWEE, NESS, *and* (E×4)(W×2)(N×4)(S×2).

Lemma 2 *The set of conformant plans of a sample is the intersection of the plans for each tag:*

$$\Pi(P_{\mathcal{B}}) = \bigcap_{v \in \mathcal{V}} \bigcap_{t \in Tags(\mathcal{B},v)} \Pi(P_{v,t}).$$

Proof: (Sketch) We prove $\Pi(P_{\mathcal{B}}) \subseteq \Pi(P_{v,t})$ by noticing that $P_{v,t}$ is essentially a relaxation of problem $P_{\mathcal{B}}$.

We now prove $\Pi(P_{\mathcal{B}}) \supseteq \bigcap_{v \in \mathcal{V}} \bigcap_{t \in Tags(\mathcal{B},v)} \Pi(P_{v,t})$. If π is invalid for $\mathcal{B}$, there is a state $q \in \mathcal{B}$ and a variable v whose value is incorrect at some point during the execution of π from q (the value contradicts either an action precondition or the goal). Let t be the restriction of q to the variables $ctx(v)$; notice that $t \in Tags(\mathcal{B}, v)$. The problem $P_{v,t}$ "simulates" the value of v during the execution of a plan. Therefore π is invalid for $P_{v,t}$. □

We write $Tags(\mathcal{B}) = \bigcup_{v \in \mathcal{V}} Tags(\mathcal{B}, v)$ the tags covered by $\mathcal{B}$ and $T = |Tags(\Phi_I)|$ the number of tags for the initial belief. Remember that $T = O(|\mathcal{V}| \times D^w)$ where D is the size of the largest domain D_v for some variable.

Theorem 3 CPCES *requires at most* 2^T *iterations.*

Proof: Consider the sample $\mathcal{B}$ at some iteration. This sample allows the candidate plan π generated at that iteration. If π is invalid, it is added to $\mho$ and all later samples $\mathcal{B}'$ will contain a state that contradicts π. Therefore from Lemma 2 $Tags(\mathcal{B}') \neq Tags(\mathcal{B})$, and no two samples during CPCES cover the same set of tags. Since there are T tags, this implies that the number of iterations is at most 2^T. □

We showed in Subsection 3.3 that some implementations of CPCES sometimes require indeed $O(2^T)$ iterations.

Theorem 4 *dp*CPCES *requires at most* $T + 1$ *iterations.*

Proof: Consider the sample $\mathcal{B}$ at some iteration and the sample $\mathcal{B}'$ at the next iteration. Because dpCPCES is dominant-preserving, we know that $\Pi(P_{\mathcal{B}'}) \subset \Pi(P_{\mathcal{B}})$. Consequently, from Lemma 2 there is a tag t covered by $\mathcal{B}'$ that was not covered by $\mathcal{B}$ or by any previous sample. So each iteration except the first one can be associated with a different tag. □

Notice that this indicates that dpCPCES requires fewer iterations than other CPCES implementations in the worst case. Yet the classical planning problems solved during dpCPCES are generally larger, which can be detrimental to the runtime. This is studied empirically in Section 5.

4.3 Samples

Theorem 5 *In general, the sample computed at the end of* CPCES *may be neither a subset nor a superset of a minimal basis. The sample computed at the end of min*CPCES *is a subset of a minimal basis. The sample computed at the end of o-* CPCES *is an optimality-basis. The sample computed at the end of o-min*CPCES *is a minimal optimality-basis.*

Proof: The call to `candidate-plan`($\mathcal{B}$) used by the general version of CPCES may produce a valid plan even if $\mathcal{B}$ is not a basis; on the same token, the procedure `counter-example-set`($\mho$) is not guaranteed to return a minimal counter-example set (for instance, when applying gCPCES to Fig. 1.a, adding $\langle 1, 0 \rangle$ to the sample guarantees that the final sample will never be minimal).

The sample computed during minCPCES at any step contains no redundant state for a subset of invalid plan. Therefore it contains no redundant state for $A^* \setminus \Pi(P)$.

The plan π returned by o- CPCES is $\arg\min \Pi(P_{\mathcal{B}})$ where $\mathcal{B}$ is the final sample. Because this plan is optimal it satisfies $\pi = \arg\min \Pi(P)$. Therefore $\mathcal{B}$ is an optimality-basis.

Finally all the invalid plans in $\mho$ computed by o-minCPCES have a lower cost than the optimal solution. Furthermore the procedure `minimise` of minCPCES removes from $\mathcal{B}$ the states that are not necessary to invalidate some element of $\mho$. Therefore the optimality-basis returned by o-minCPCES is also minimal. □

5 Experiments

This section reports an experimental analysis studying the practical implications of the different sampling configurations of CPCES. Each sampling strategy has potential effect on both efficiency of the planning process and quality of the produced solutions. We measure efficiency by considering run-time and total number of iterations necessary to find a solution, or to prove its absence. Then we measure quality differently depending on whether the instance is solvable or not. For solvable instances, solution quality is the length of the plan. For unsolvable instances the quality of the strategy is given by the size of the counter-examples set justifying the non-existence of a solution. We expect the greedy policy to outperform the other strategies in terms of iterations (this is in fact the only dominance-preserving strategy), but at the price of compromising solution quality. The first point has implications on run-time, but it is not clear how much. As a matter of facts, while fewer iterations can translate in faster convergence, each iteration may need to deal with classical planning instances of non-negligible size. This experimental analysis is aimed at studying this relation from an empirical standpoint. For comparison reasons, we also use the T1 planner (Albore, Ramírez, and Geffner 2011), which is one of the state of the art available conformant planner.

Our benchmark suite includes instances presented in Grastien and Scala (2017a) that feature a balanced mix of problems with width = 1 and with width > 1. To this set we added two variants of the domain presented in Section 1; both variants feature a swamp somewhere in the grid but differ from the possible presence of a wall within the grid. This domain has been designed to stress problems with a non-trivial width (instances of this domain have all width equals to 2), which make them challenging in the conformant planning literature; in particular for planners which make direct use of the width either in the heuristic (Albore, Ramírez, and Geffner 2011) or in the reduction to classical planning (Palacios and Geffner 2007). For both variants we generate 18 instances, which scale on the size of the grid (from 3×3 to 20×20); the swamp is chosen randomly in the grid. We

refer to such two variants EMPTYGRID and WALLGRID, respectively. Note that instances from EMPTYGRID are solvable only when the swamp is next to one of the border of the grid. In the WALLGRID instances, we make sure that the swamp is always surrounded by at least one wall (which can be either the border of the grid, or a wall positioned on the immediate left side). Instances from WALLGRID are then all solvable.

For the experimental setting we used z3 (de Moura and Bjørner 2008) to find a counter-example for the validity of the candidate plan. In order to solve the classical planning problem we use FF-V2.3 (Hoffmann and Nebel 2001) with the reduction presented in Grastien and Scala (2017a). We also tried other planners, yet FF-V2.3 was the fastest in general, especially in our context where we need a planner with quick startup. The remaining part of architecture is implemented in Java 8, while the heuristic in hCPCES is obtained using a modified version of FF-V2.3. The orchestration among the various modules is obtained using the SMTLIB and PDDL languages respectively for interacting with the z3 solver and FF-V2.3. Computation ran on an Xeon@3.4 GHz with 4800 secs and 4GB memory cut-off[4].

5.1 Results for the IPC Domains

Table 3 presents experimental data collected for the IPC domains. It measures, on a per domain basis, the run-time, number of samples generated at the end of the iterations, and number of iterations for each of the presented sampling strategies: gCPCES, hCPCES, refCPCES.

The greedy sampling is the only proved to be a dominance-preserving strategy and because of that, it generally converges faster towards the solution. However, both hCPCES and refCPCES compute much smaller samples in the end. hCPCES substantially improves on the number of iterations required by refCPCES, whilst keeping the number of samples generally way lower than gCPCES. However, in these domains, only in few instance this translates to an overall run-time lower than gCPCES. The benchmark suite also includes two unsolvable instances; one for BLOCK, the other for RAOSKEY. refCPCES terminates with a sample of size one for both instances. hCPCES also only finds one counter-example for RAOSKEYS, but finds instead two counter-examples for the unsolvable instance of BLOCK. Another interesting phenomena to be observed is the quality of the resulting plans for each of the tested strategy. refCPCES finds shorter plans in general than gCPCES; this also applies for hCPCES, except for UTS. Note that none of the configurations use an optimal planner for the generation phase. Yet it seems that the quality of the plans is statistically correlated to the size of the sample. refCPCES, in LOOKANDGRAB, decreased the average plan length by 20%. This is however not completely surprising, as it is expected from a suboptimal planner to greedily select longer plans in addressing a larger number of initial states. On the other hand, it can remain closer to optimal if only a subset of them needs to be addressed.

[4]The software, the instances and the experimental data are available at https://bitbucket.org/enricode/cpces.

	Run Time			Iterations			Samples			Plan Length			
I	G	H	R	G	H	R	G	H	R	G	H	R	T1
9	**0.23**	0.34	0.62	**3**	**3**	4	**2**	**2**	**2**	10	10	10	0.05
16	0.32	**0.30**	0.41	**5**	**5**	11	4	**2**	**2**	18	18	18	0.05
25	**0.29**	0.73	0.55	**5**	**5**	19	4	**3**	4	20	20	20	0.05
36	**0.29**	0.81	0.41	**4**	11	9	3	4	**2**	24	24	24	0.05
256	**4.69**	25.45	11.54	**26**	135	119	26	11	**8**				F
289	11.32	**10.83**	21.16	**39**	59	193	39	8	**7**				F
324	**6.88**	14.80	16.27	**28**	72	138	28	10	**8**				F
361	46.29	46.53	**10.82**	**54**	159	96	54	9	**4**				F
400	733.20	28.81	**14.07**	138	109	**93**	138	**9**	10				F

(a) EMPTYGRID

	Run Time			Iterations			Samples			Plan Length			
I	G	H	R	G	H	R	G	H	R	G	H	R	T1
144	**7.49**	123.27	92.99	**54**	633	818	53	**25**	26	76	76	76	F
169	**31.71**	221.85	159.80	**89**	807	1021	88	**27**	28	80	80	80	F
196	**46.56**	348.15	276.65	**89**	1066	1156	88	**29**	32	88	88	88	F
225	**82.32**	464.67	439.47	**104**	1148	1437	103	**32**	35	94	94	94	F
256	**90.04**	808.90	679.39	**93**	1519	1588	92	**35**	37	102	102	102	F
289	**117.26**	2396.05	925.68	**98**	2674	1957	97	42	**37**	108	108	108	F
324	**210.20**	2829.33	2381.65	**115**	2911	2791	114	**40**	45	114	114	114	F
361	**667.82**	F	F	**146**	F	F	**145**	F	F	120	F	F	F
400	**1497.59**	F	F	**185**	F	F	**184**	F	F	126	F	F	F

(b) WALLGRID

Table 2: Selection of EMPTYGRID and WALLGRID instances. I indicates the size of the grid. G stands for the Greedy, H for Heuristic and R for Refined sampling strategy. The right most column reports runtime for T1

In Table 4, we report, for each domain, the largest instances solved by all, where we contrast time spent for plan generation versus time spent for sampling. It turned out that the overhead for the sampling in hCPCES is large (e.g, the instance selected for BOMB), but sometimes can fasten plan generation. For instance in DISPOSE-ONE, hCPCES takes 20.36 seconds for plan generation in 462 iterations, generating a plan which is half the plan generated by gCPCES. Other times however hCPCES and gCPCES end with a very similar size sample, and so the effort spent by hCPCES to reason over the counter-example set is only detrimental against the overall run-time.

Another interesting aspect is the comparison between the size of the samples generated by hCPCES and refCPCES. Although hCPCES is not guaranteed to find minimal bases, in many cases it generates a sample that is of size comparable to the one generated by refCPCES. This empirically shows that the employed heuristic estimate provide a good trade-off between the size of the sample, and the number of iterations.

5.2 Results for the Grid-Based Domains

Table 2 shows a selection of 10 out of 18 instances for the two grid based domains. We took the largest solvable and the unsolvable instances (if applicable). In EMPTYGRID, all the instances larger or equal to 256 cells (16×16) are not solvable: the swamp is not surrounded by any wall.

EMPTYGRID. Unexpectedly, all the sampling strategies lead to the same plan length, but differ on the size of the samples generated, sometimes considerably. When the differences

	Coverage			Run Time			Samples			Iterations			Plan Length			
Domain	G	H	R	G	H	R	G	H	R	G	H	R	G	H	R	T1(Cov.)
BOMB(9)	8	3	0	**1.2**	140.1	NA	26.3	**25.3**	NA	**27.3**	197.3	NA	40	40	NA	**9**
COINS(9)	8	8	1	**0.80**	2	23.40	16	6	**5**	17	**16**	348	31	30	**29**	**9**
DISPOSE(11)	5	3	0	**9**	17.90	NA	30	**27.70**	NA	31	**30**	NA	72	**69.70**	NA	**8**
UTS(16)	**15**	11	6	**0.60**	4.50	171.10	9.50	**7**	**7**	10.50	**9.50**	919	**19**	19.50	**19**	14
BLOCKS(3)	**3**	**3**	2	**0.30**	20.10	1.10	4.50	3.50	**2**	5.50	**4.50**	14	12.50	11.50	**10**	2
BLOCKU(1)	1	1	1	**0.43**	2.78	6.67	5	2	**1**	**5**	6	121	NA	NA	NA	0
LOOKANDGRAB(18)	18	18	18	**20**	35.50	217.70	8.50	6.60	**3.70**	**9.50**	13.30	283.10	49.80	46.70	**36.30**	15
DISPOSE-ONE10	**5**	**5**	2	**0.60**	3	19.60	14	**5**	5.50	15	**14**	290.50	34.50	**31**	33	4
RAOSKEYS(2)	2	2	2	**0.70**	2.10	1870.80	13	9.50	**8.50**	14	**12.50**	6931	36	34.50	**34**	1
RAOSKEYUS(1)	1	1	1	**0.61**	3.21	1.67	9	**1**	**1**	**9**	12	46	NA	NA	NA	0

Table 3: Overview of the results on the IPC domains, track on uncertainty. G, H, R stands for gCPCES, hCPCES and refCPCES. Last column shows coverage for T1. Bold highlights best result.

	Generation Time			Sampling Time			Samples Size			Iterations			Plan Length		
Problem	G	H	R	G	H	R	G	H	R	G	H	R	G	H	R
DISPOSE(P4.3)	**1.95**	2.89	NA	**3.18**	21.83	NA	40	**36**	NA	41	40	NA	94	**87**	NA
BLOCKU(P4)	0.07	**0.06**	2.11	**0.14**	0.37	3.11	5	2	**1**	5	6	121	NA	NA	NA
RAOSKEYUS(P4)	0.12	**0.11**	0.34	**0.33**	2.44	0.85	9	**1**	**1**	9	12	46	NA	NA	NA
DISPOSE-ONE(P2.3)	0.25	**0.16**	11.68	**0.38**	1.86	13.59	19	**6**	**6**	20	14	502	45	**38**	40
DISPOSE-ONE(P4.2)	2093.55	**20.36**	NA	**9.24**	272.26	NA	75	**30**	NA	76	462	NA	205	**111**	NA
LOOKANDGRAB(P8_3_3)	6.25	4.54	**4.43**	**1.33**	11.39	3.90	10	7	**4**	11	16	34	60	58	**42**
BOMB(P20-10)	**0.65**	13.29	NA	**0.67**	374.50	NA	31	**30**	NA	32	533	NA	40	40	NA
UTS(P6)	**0.18**	0.26	38.42	**0.35**	2.45	902.71	15	**12**	**12**	16	14	4114	**34**	37	**34**
COINS(P10)	0.16	**0.12**	8.55	**0.26**	1.52	7.25	16	6	**5**	17	16	348	31	30	**29**

Table 4: A per instance based analysis contrasting plan generation and sampling time over the proposed strategies. Times here are cumulative. Bold highlights best result when significant for our evaluation.

are negligible, so is the run-time. When the sizes are instead very different, and this happens for the last 5 instances, the strategies with smaller samples (in this case hCPCES and refCPCES) lead to improvement of the performances to the point to be even more efficient than gCPCES. This happens in particular for instance 361 and instance 400.[5] Remarkable are the performances of refCPCES which does not seem to be affected by the size of the grid for the unsolvable instances, and can quickly find a minimal counter-examples explaining the unsolvability of the task. Let us recall that, in this case, the classical planner is the component responsible for proving the unsolvability. The classical planning problem is much smaller and has a more representative initial state when generated by either hCPCES or refCPCES.

WALLGRID. In this case all the strategies lead to the same plan too, but the trend is different from that of EMPTY-GRID. gCPCES outperforms both hCPCES and refCPCES in terms of run-time. For both hCPCES and refCPCES, the size of the sample generated at the end of the computation is in fact on the average only half the size of that generated by gCPCES. Interestingly, and unexpectedly, hCPCES sampling leads to samples that are even smaller than the ones generated by refCPCES. Let us remember the reader that refCPCES is aimed at finding minimal, not minimum, counter-examples set.

From our experiments T1 managed to solve (very quickly) only the small instances for both domains, timing out on all the instances larger than 36 cells. This further motivates our interest in looking at methods in the CPCES framework.

[5] Strategies on much larger instances (till 900 cells) highlighted speed-ups till 2 orders of magnitude (refCPCES vs gCPCES).

6 Conclusion

We study the properties of CPCES, a recent conformant planner, with respect to known concepts in the literature: bases, tags, and width. We propose different variants of CPCES and study their performance (runtime, quality of output) from a theoretical and an empirical perspective. As a result of this investigation we can bound the number of iterations of CPCES using the number of tags needed to solve a problem. This holds as long as we use a strategy that satisfies the property of being dominance-preserving. We also identify another class of sampling which instead guarantees the discovery of minimal basis, and a heuristic-based sampling strategy providing a middle ground between these two extremes. A spin-off of this work is a different characterisation of the plan generation phase. We expect that an *incremental* plan generation can provide substantial speed up, especially in problems which exhibit clear decompositions.

Acknowledgements

The authors want to thank the reviewers and Patrik Haslum for their interesting feedback.

References

Albore, A.; Ramírez, M.; and Geffner, H. 2011. Effective heuristics and belief tracking for planning with incomplete information. In *ICAPS*.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Bonet, B., and Geffner, H. 2014. Belief tracking for planning with sensing: width, complexity and approximations. *Journal of Artificial Intelligence Research (JAIR)* 50:923–970.

Bylander, T. 1991. Complexity results for planning. In *Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, 274–279.

Castellini, C.; Giunchiglia, E.; and Tacchella, A. 2003. Sat-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artificial Intelligence* 147(1-2):85–117.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence* 159(1-2):127–206.

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.

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

Grastien, A., and Scala, E. 2017b. Verifying the validity of a conformant plan is co-NP-complete. http://vixra.org/abs/1705.0340.

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.

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

Palacios, H., and Geffner, H. 2007. From conformant into classical planning: Efficient translations that may be complete too. In *ICAPS*, 264–271.

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

Rintanen, J. 2004. Complexity of planning with partial observability. In *ICAPS*, 345–354.

Smith, D. E., and Weld, D. S. 1998. Conformant graphplan. In *Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, (AAAI/IAAI-98)*, 889–896.

To, S. T.; Pontelli, E.; and Son, T. C. 2009. A conformant planner with explicit disjunctive representation of belief states. In *ICAPS*.