Computing Superior Counter-Examples for Conformant Planning

Xiaodi Zhang,[1] **Alban Grastien,**[1] **Enrico Scala**[2]
[1]Research School of Computer Science, Australian National University, Canberra
[2]Università degli Studi di Brescia
xiaodiz@outlook.com, alban.grastien@anu.edu.au, enricos83@gmail.com

Abstract

In a counter-example based approach to conformant planning, choosing the right counter-example can improve performance. We formalise this observation by introducing the notion of "superiority" of a counter-example over another one, that holds whenever the superior counter-example exhibits more *tags* than the latter. We provide a theoretical explanation that supports the strategy of searching for maximally superior counter-examples, and we show how this strategy can be implemented. The empirical experiments validate our approach.

1 Introduction

Conformant planning is the problem of finding a robust plan despite uncertainty in the initial state (Smith and Weld 1998). This problem is EXPSPACE-COMPLETE (Haslum and Jonsson 1999). One of the aspects that make it hard is the fact that the number of possible initial states is exponential in the number of state variables; this exponential burden makes brute-force enumeration impractical.

We are interested in a recent approach proposed by Grastien and Scala (2017; 2018) and dubbed GCPCES, that addresses this specific issue. In GCPCES, a *candidate plan* is computed based on a small number of initial states (the *sample*) that are assumed to be representative of the planning problem. The validity of this plan for the complete problem is then tested. If the plan is found invalid, a *counter-example* (i.e., an initial state for which the plan is invalid) is generated and added to the sample, since this counter-example is clearly representative of some relevant aspect of the planning problem. This process is repeated until a conformant plan is discovered or it is proved that the problem yields no solution. Notwithstanding its simplicity, this counter-example guided search works well in practice since it generates only relevant counter-examples.

In the general framework, a large freedom is left to GCPCES for choosing the counter-example. This freedom however can bring the planner to enrich the sample in a manifestly suboptimal way, as we illustrate in Section 3. There are, instead, good choices that can make the planner converge much faster. In this paper we show how some counter-examples are more informative than others. We argue that selecting such counter-examples reduces the number of iterations required to find a valid plan, which leads to improved performance and more succinct explanations when the sample is used as a justification for the conformant plan.

The goal of this work is to study why some counter-examples are better than other ones, and how such better counter-examples can be generated. The paper defines *superiority* between counter-examples by exploiting the notion of *tags*, a well-known concept in conformant planning that can be interpreted as a "contingency" that the planner must address. A counter-example is considered superior to another one if the former exhibits more new tags than the latter (compared to the tags already covered by the current sample). The paper provides a SAT-based procedure that targets the computation of optimal counter-examples under this criterion, and, through an empirical evaluation, studies the implications of using it in an extended version of GCPCES. For problems that can be decomposed (those where a state covers several tags at once) this technique improves performance greatly. This not only extends the applicability of GCPCES to a larger class of problems, but also, and more importantly, shows how to exploit problem structure in a non-heuristic search framework. As we show in the experiments, this allows us to approach the performance of heuristic search planners that greatly exploit the problem structure, and at the same time to leverage from the learning power of a counter-example guided refinement technique that proves particularly beneficial in highly constrained problems such as conformant planning.

Section 2 provides the technical background necessary to understand this work. Section 3 introduces the notion of superiority between counter-examples, first with an example and then formally. Section 4 shows how the procedure for generating a counter-example can be adapted to guarantee that that counter-example is optimal. Section 5 discusses related work. Section 6 presents an empirical evaluation.

2 Background

2.1 Conformant Planning

We are interested in the problem of deterministic conformant planning, which we present now. We reuse the notation from

Grastien and Scala (2017).

We write $\mathcal{V}$ the set of *state variables*. Each variable $v \in \mathcal{V}$ is associated with a *domain* D_v of values. A *state* s is a function that assigns each variable v to a value $s(v) \in D_v$.

An *action* a is a pair $\langle prec, eff \rangle$ where *prec* is the *precondition* and *eff* is the *conditional effect function*. The action is *applicable* in a state if the precondition *prec* evaluates to *true* in that state. The conditional effect function *eff* associates each pair $\langle v, \ell \rangle$ with a condition, so that the application of the action assigns v to ℓ whenever *eff*(v, ℓ) evaluates to *true* in the current state. Formally, the *application* of action a in state s leads to the state $s' = s[a]$ where $s'(v) = \ell$ iff

- either *eff*(v, ℓ) evaluates to *true* in the current state s (we assume that *eff*(v, ℓ_1) and *eff*(v, ℓ_2) cannot both evaluate to *true* for the same variable v and two values $\ell_1 \neq \ell_2$);
- or variable v is already assigned to ℓ ($s(v) = \ell$) and *eff*(v, ℓ') evaluates to *false* for all values $\ell' \in D_v$.

A *plan* $\pi = a_1 \dots a_n$ is a sequence of actions. It is *applicable* in state s if all of its actions a_i are applicable in $s[a_1] \dots [a_{i-1}]$. It leads to state $s[a_1] \dots [a_n]$.

A *belief state* $\mathcal{B}$ is a propositional formula defined over *assignments* of the form $(v = \ell)$. $\mathcal{B}$ represents implicitly the set of states that make the formula evaluate to *true*. We use the formula and the set of states interchangeably.

A *(deterministic) conformant planning problem* is a tuple $P = \langle \mathcal{V}, A, \Phi_I, \Phi_G \rangle$ where $\mathcal{V}$ is the set of state variables, A is the set of actions, Φ_I is the *initial belief state*, and Φ_G is the *goal belief state*. The plan π is *valid* for state s if it is applicable in this state and this application leads the system to a goal state ($s[\pi] \in \Phi_G$). The plan π is *valid* for problem P if it is valid for all initial states.

The purpose of the conformant planning problem is to compute a valid plan for this problem, or to determine that no such a plan exists.

2.2 gCPCES

We build our work on top of the conformant planner gCPCES (Grastien and Scala 2018). Classical planning problems are a special class of conformant planning problems where Φ_I includes only one initial state. Classical problems are easier to solve both in the computational theory sense and in practice. Indeed, a number of effective domain independent heuristics have been proposed in the literature, and modern planners scale nowadays reasonably well with respect to problem size (Vallati et al. 2015). A conformant problem P can be translated into a classical one, but the resulting problem has a size linear in the number of initial states, which is often intractable.

gCPCES (Algorithm 1) addresses this issue by replacing the initial belief state with a small number of initial states called the *sample* $\mathcal{B}$, and using the sample to propose a *candidate plan* π, i.e., a plan valid for the sample; if no such plan exists then the conformant planning problem is unsolvable. Then it searches for a *counter-example*, i.e., an initial state for which π is invalid. If no such state exists, the plan is valid; otherwise the state is added to the sample and a new iteration starts. The procedures that produce a candidate plan and that generate a counter-example are described by Grastien and Scala (2017).

Algorithm 1 The conformant planner gCPCES.

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

We write $\Pi(P)$ the set of valid plans for the problem P, and, for a subset of initial states ($\mathcal{B} \subseteq \Phi_I$), we write $\Pi(\mathcal{B})$ the set of valid plans when Φ_I is replaced with $\mathcal{B}$. gCPCES relies on the property $\mathcal{B}_1 \subseteq \mathcal{B}_2 \Rightarrow \Pi(\mathcal{B}_1) \supseteq \Pi(\mathcal{B}_2)$. Therefore adding elements to the sample reduces the set of valid plans of this sample so that, eventually, $\Pi(\mathcal{B})$ equals $\Pi(P)$. This property guarantees that no plan will be missed by gCPCES.

Performance-wise, gCPCES has proved to be competitive with alternative conformant planning approaches, in particular for problems that have a non-trivial width (cf. Subsection 3.3).

3 What is a Good Counter-Example

3.1 Example

We use the planning domain DISPOSE to illustrate the shortcoming of gCPCES that we address in this paper. This domain asks a robot to collect a number of items scattered around on a map and then drop them in a specific "trash-at" location. The initial location of the items is unknown. The robot has access to three actions: PICK-UP an item in its current location (the robot will then hold the item if it is in the location—it can hold several items at the same time); MOVE from one location to a neighbouring location; and DROP an item in the trash-at location. The solution to the problem is for the robot to visit all locations, in each location try to pick up each item, then move to the trash-at location and dispose of all items.

For an instance of the DISPOSE domain with 3 items and 4 locations (A to D), we illustrate on Table 1 a "bad" execution and a "good" execution of gCPCES, where the assessment is defined by how fast gCPCES finds a valid plan; so the second execution is "good" because it involves only four iterations against the ten iterations of the "bad" execution. In both cases we assume that the candidate plan produced by the planner is optimal for the current sample.

The first execution appears on the top array of Table 1. The initial candidate plan π_0 is always the empty plan (perform no action). The first counter-example generated to invalidate the initial plan is represented by first column of the table, and it assumes that all items start in location A. Consequently the next candidate plan π_1 will be: go to location

#	1	2	3	4	5	6	7	8	9	10
L_1	A	B	A	A	C	A	A	D	A	A
L_2	A	A	B	A	A	C	A	A	D	A
L_3	A	A	A	B	A	A	C	A	A	D

#	1	2	3	4
L_1	A	B	C	D
L_2	A	B	D	C
L_3	A	B	C	D

Table 1: Bad and good executions of gCPCES: each column describes a counter-example generated by gCPCES at a given iteration; L_i is the initial location of the ith item.

A, pick up all items, go to the trash-at location, and drop all items. The second counter-example (second column) then assumes that the first item is in location B while the second or third items in location A. Therefore the next plan π_2 expands π_1 by asking the robot to visit location B and try to pick up the first item from this location; however the robot will not be instructed to pick up the second or third items from location B (this is unnecessary given the current sample). Hence the third counter-example assumes that the second item starts in location B, and so on.

Comparatively the good execution is illustrated on the bottom array in Table 1. In this execution the second counter-example assumes that B is not only the starting location of the first item, but also that of the second and third items. We see that the execution requires fewer iterations as only four counter-examples are necessary to converge. (Notice also that the items do not need to start in the same location in each counter-example, as illustrated with the third and fourth counter-example.)

3.2 Analysis

What the example of Table 1 illustrates is that we want the counter-examples generated by gCPCES to be as different from one another as possible, because similar counter-examples highlight the same aspects of the planning problem. This raises the question of what "different" means.

One idea would be to maximise the number of assignments that the counter-examples exhibit. There is no guarantee however that this solution will be satisfactory. Consider for instance a planning problem where the state variables have a binary (0/1) domain. Assume that the first counter-example associates each state variable with 0 and the second counter-example associates each state variable with 1. Then all possible assignments are covered with these two counter-examples and it is unclear what the next counter-example should look like. One could try to maximise the difference between the new counter-example and all existing counter-examples, for example having precisely half the variables associated with 0. Once again however there is no real guarantee that this solution is particularly good. It is very easy to come up with examples where this approach will fail.

Fundamentally using the variable assignment is flawed because it shallowly consider only the *syntactical* aspects of the problem description. Instead we propose and study a more profound measure of similarity that is more concerned with the *semantics* of the conformant planning problem at hand. Interestingly, this measure can be grounded on the notion of tags, a well known notion identified in literature that targets at possible ways to decompose the problem.

3.3 Tags and Conformant Planning

We now recall known definitions that relate tags (Palacios and Geffner 2009) to conformant planning problems. A *subgoal* φ is a conjunct of an action precondition or the goal; in other words, action preconditions and the goal are conjunctions of subgoals. Importantly a subgoal is a logic formula that may be required to evaluate to *true* at some point of the execution for a plan to be valid.

A variable v *depends* on another variable v' if there exists an action $a = \langle \mathit{prec}, \mathit{eff} \rangle$ and a value $\ell \in D_v$ such that the conditional effect $\mathit{eff}(v, \ell)$ mentions variable v'. The *context* $\mathit{ctx}(\varphi)$ of subgoal φ is the set of variables mentioned by φ, the variables they depend on and, by transitivity, all the variables that they depend on. So, consider the DISPOSE domain; the action PICK-UP-1-A (which picks up the first item if it currently sits in location A) has precondition $L_{\mathrm{R}} = A$ and conditional effect: if $L_1 = A$ then $\mathit{holding}_1 = \top$. This action defines a dependency between variable $\mathit{holding}_1$ and L_1, which can be interpreted as follows: the value of L_1 at any time during execution (and, in particular, initially) may affect the value of $\mathit{holding}_1$ later in the execution. Notice however that the preconditions are not used in the definition of dependency: in DISPOSE, $\mathit{holding}_1$ does not depend on L_{R}. Preconditions are handled through subgoals.

We write C the set of contexts of the conformant planning problem. The *width* $w(c)$ of a context c is the number of state variables of c that are uncertain in the initial state. The *width* $w(P)$ of the conformant planning problem P is the largest width of some context of P: $w(P) = \max_{c \in C} \; w(c)$.

Given a context c and a state q, the *tag* (Palacios and Geffner 2009) of q for c is the restriction of the state q to the variables in c and is denoted $\mathit{tag}_c(q)$. Given a belief state $\mathcal{B}$ and a context c, the set of tags of $\mathcal{B}$ for c is the set $T_c(\mathcal{B}) = \{\mathit{tag}_c(q) \mid q \in \mathcal{B}\}$. We write $T(\mathcal{B}) = \bigcup_{c \in C} \; T_c(\mathcal{B})$ the set of tags of the belief $\mathcal{B}$ for all contexts.

It is possible (Palacios and Geffner 2009) to associate each tag t with a set $\Pi(t)$ of plans so that for all belief state $\mathcal{B}$, the following statement holds:

$$\Pi(\mathcal{B}) = \bigcap_{t \in T(\mathcal{B})} \Pi(t). \tag{1}$$

At this point we notice that if a context c is a subset of another context c', then the information in the tag $\mathit{tag}_c(q)$ is included in the tag $\mathit{tag}_{c'}(q)$ (since $\mathit{tag}_c(q)$ is the restriction of $\mathit{tag}_{c'}(q)$ to the variables in c). Therefore it is possible to narrow C down to those contexts that are superset maximal.

There is an interesting connection between tags and gCPCES. Indeed, the equality (1) holds both when $\mathcal{B}$ is the initial belief state or a sample. In particular if the sample $\mathcal{B}$ is such that $T(\mathcal{B})$ equals $T(\Phi_I)$, then $\Pi(\mathcal{B}) = \Pi(P)$ and the candidate plan produced for $\mathcal{B}$ will be valid for P. Notice that the equality of the sets of tags is a sufficient condition but not necessary. For instance, some tags can be redundant

(i.e., some tag t' satisfies $\Pi(t') \supseteq \bigcap_{t \in T(\mathcal{B}) \setminus \{t'\}} \Pi(t)$). Furthermore gCPCES's procedure aimed at producing the candidate plan can be lucky and come up with a plan that is valid for the conformant problem before the equality holds.

Whenever a candidate plan π is proved invalid, the counter-example q that is generated is such that it exhibits a new tag $t \notin T(\mathcal{B})$ that disproves this plan (i.e., $\pi \notin \Pi(t)$). Completeness of gCPCES is guaranteed by the convergence of $T(\mathcal{B})$ towards $T(P)$. Maximising the number of new tags can accelerate convergence. We formalise this notion next.

3.4 Superior Counter-Examples

We exploit tags by introducing the notion of superiority of counter-examples given a current belief state.

Definition 1. *Given a belief state $\mathcal{B}$, counter-example q' is* strictly superior *to counter-example q, denoted $q' \succ_{\mathcal{B}} q$, if the following holds:*

$$T(\mathcal{B} \cup \{q'\}) \supset T(\mathcal{B} \cup \{q\}).$$

We use the notation $q' \succeq_{\mathcal{B}} q$ and we say that q' is superior to q when the relation is not strict ($T(\mathcal{B} \cup \{q'\}) \supseteq T(\mathcal{B} \cup \{q\})$). The naming "superior" is based on the following two lemmas that show that a superior counter-example makes it more likely to find a valid plan and thus reduces the number of iterations of gCPCES.

Lemma 1. *If q' is superior to q for belief state $\mathcal{B}$ then the property $\Pi(\mathcal{B} \cup \{q'\}) \subseteq \Pi(\mathcal{B} \cup \{q\})$ holds.*

In other words the counter-example q' is such that the belief state $\mathcal{B} \cup \{q'\}$ *dominates* $\mathcal{B} \cup \{q\}$ (Grastien and Scala 2018).

Lemma 2. *If q' is superior to q for belief state $\mathcal{B}$ then for all set of states $\mathcal{B}'$, q' is superior to q for belief state $\mathcal{B} \cup \mathcal{B}'$.*

Proof Sketch. This result relies on the decomposability of $T(\cdot)$ wrt. the set of states: $T(\mathcal{B} \cup \{q\} \cup \mathcal{B}') = T(\mathcal{B} \cup \{q\}) \cup T(\mathcal{B}') \subseteq T(\mathcal{B} \cup \{q'\}) \cup T(\mathcal{B}') = T(\mathcal{B} \cup \{q'\} \cup \mathcal{B}')$. □

Lemma 1 shows that superiority increases the chances of finding a valid plan. Lemma 2 shows that superiority is carried over as more counter-examples are generated, which means that there is no regret associated with choosing a counter-example q' that is superior to q in the current situation (i.e., when $\mathcal{B}$ is known while $\mathcal{B}'$ is not).

4 Computing Good Counter-Examples

In this section we show how to compute an optimal (maximally superior) counter-example. Deciding whether a plan is valid for a conformant planning problem (and thereby finding a counter-example to the plan) is CO-NP-COMPLETE (Grastien and Scala 2018); consequently this problem has been solved with SAT techniques. Finding an optimal counter-example can be reduced to a MAXSAT problem that maximises the *number* of new tags covered. The solution set for this reduction, however, is unnecessarily too narrow: the set of new tags of an optimal counter-example is not always cardinality maximum, merely subset maximal. For this reason, we propose instead to start from any counter-example and improve it until it is maximally superior.

Algorithm 2 `compute-optimal-counter-example`: ($\perp$ indicates that there is no solution).

```
input: candidate plan π, current sample B
q := generate-counter-example(π)
if q = ⊥ then
   return π is valid
loop
   q' := improve-counter-example(B, q)
   if q' = ⊥ then
      return q
   q := q'
```

4.1 The General Improvement Algorithm

The algorithm for computing an optimal counter-example works in an iterative fashion. This is expressed in Algo. 2, which replaces `compute-counter-example` of Algo. 1. The procedure first searches for a counter-example. If there is no such counter-example, the plan is valid. Otherwise the algorithm incrementally improves the counter-example, i.e., searches for a counter-example that is strictly superior to the current counter-example. The subroutines `generate-counter-example` and `improve-counter-example` are described in the next subsections.

The procedure of Algo. 2 is guaranteed to terminate for the following reason: the state q' computed at some iteration is strictly superior to q. This implies that q' exhibits more new tags than q: $T(q') \setminus T(\mathcal{B}) \supset T(q) \setminus T(\mathcal{B})$. The number of tags of a state is bounded by the number of subgoals; therefore the termination is ensured.

For invalid plans, notice that Algo. 2 always ends exactly when the subroutine `improve-counter-example` terminates with no counter-example. This differs from the original implementation of gCPCES for which finding no counter-example happens only once, when the plan is valid. This may become an issue in situations where showing that no better counter-example exist is hard.

4.2 Computing a Counter-Example

We now show how a counter-example can be computed (Grastien and Scala 2017) using SAT, as we use a similar method to find a strictly superior counter-example.

Assume the plan is $\pi = a_1 \dots a_n$. Finding a counter-example consists in finding an initial state $q_0 \in I$ such that executing the plan makes the system go through the states $q_1, \dots, q_n$ and either one of the actions is not applicable or the goal is not satisfied in the final state q_n.

In practice for every state variable $v \in \mathcal{V}$, for every value $\ell \in D_v$ in the domain of v, and for every timestep $i \in \{0, \dots, n\}$, a SAT variable $v^{i,\ell}$ is defined that evaluates to *true* iff the value of the variable v in the state q_i is ℓ. The following three constraints are then defined:

1. ϕ_I encodes that q_0 is an initial state;
2. ϕ_A encodes that each state q_i is the result of applying a_i in state q_{i-1} (ignoring whether the action is applicable);

3. ϕ_{nval} encodes the fact that either one of the actions is inapplicable or the goal is not reached in q_n.

The definition of these constraints is fairly standard and we do not detail them here.

ϕ_π is the formula $\phi_I \wedge \phi_A \wedge \phi_{nval}$. There is a counter-example to plan π iff ϕ_π is satisfiable and a counter-example is represented by the variables $v^{i,\ell}$ for $i = 0$.

4.3 Computing a Better Counter-Example

We now discuss how, given a sample $\mathcal{B}$ and a counter-example q, a strictly superior counter-example q' can be computed. As for the previous subsection, we achieve this by taking a model (if any), and hence a counter-example from the following SAT formula, namely, $\phi_{\mathcal{B},q}$:

$$\phi_I \wedge \phi_{\succeq_\mathcal{B}} \wedge \phi_{\not\preceq_\mathcal{B}}$$

where $\phi_{\succeq_\mathcal{B}}$ and $\phi_{\not\preceq_\mathcal{B}}$ are defined next. Intuitively $\phi_{\succeq_\mathcal{B}}$ specifies that the new tags of q will be kept (formally $q' \succeq_\mathcal{B} q$); $\phi_{\not\preceq_\mathcal{B}}$ specified that q' exhibits at least one extra new tag (formally $q' \not\preceq_\mathcal{B} q$).

We write $C_{new}(\mathcal{B}, q)$ the set of contexts for which the tag of q is not a tag of $\mathcal{B}$. Formally:

$$C_{new}(\mathcal{B}, q) = \{c \in C \mid tag_c(q) \notin T_c(\mathcal{B})\}.$$

We then define $\phi_{\succeq_\mathcal{B}}$ as

$$\bigwedge_{c \in C_{new}(\mathcal{B},q)} tag_c(q)[\mathcal{V}/\mathcal{V}_0]$$

where $t[\mathcal{V}/\mathcal{V}_0]$ is the conjunction of literal $v^{0,\ell}$ for each assignment $v \to \ell$ of a tag t. For instance, if t is $\{v_1 \to \ell_1, v_2 \to \ell_2, v_3 \to \ell_3\}$ then $t[\mathcal{V}/\mathcal{V}_0]$ is $v_1^{0,\ell_1} \wedge v_2^{0,\ell_2} \wedge v_3^{0,\ell_3}$. This constraint essentially encodes the fact that all the new tags of the current counter-example should also be covered by the superior counter-example.

Furthermore we define $\phi_{\not\preceq_\mathcal{B}}$ as

$$\bigvee_{c \in C \setminus C_{new}(\mathcal{B},q)} \neg \bigvee_{t \in T_c(\mathcal{B})} t[\mathcal{V}/\mathcal{V}_0].$$

This constraint encodes the fact that there should be a context for which the current counter-example does not exhibit a new tag, while the superior counter-example does.

Lemma 3. *There exists a state q' that is strictly superior to q for $\mathcal{B}$ iff $\phi_{\mathcal{B},q}$ is satisfiable, and one such state is represented by the variables $v^{i,\ell}$ where $i = 0$.*

Essentially the lemma relies on the following observations. First if q' is superior to q for $\mathcal{B}$, then it must exhibit the new tags $T(q) \setminus T(\mathcal{B})$ of q (compared to $\mathcal{B}$); this is achieved by the formula $\phi_{\succeq_\mathcal{B}}$.

Second if q' is strictly superior to q for $\mathcal{B}$, then it must exhibit at least one extra tag. This is achieved by the formula $\phi_{\not\preceq_\mathcal{B}}$ which we explain now. Since we ask q' to exhibit the same new tags as q the extra tag must come from a context outside of $C_{new}(\mathcal{B}, q)$. Given a context c the tag $t' = tag_c(q')$ of q' associated with c is new and t' is not in $T_c(\mathcal{B})$. Therefore one of the extra tags of q' is associated with c iff $\neg \bigvee_{t \in T_c(\mathcal{B})} t[\mathcal{V}/\mathcal{V}_0]$ holds. Hence, the definition $\phi_{\not\preceq_\mathcal{B}}$ guarantees that q' exhibits an extra tag.

Finally notice that the definition of $\phi_{\mathcal{B},q}$ is accurate, i.e., it only excludes the states that are not initial or not strictly superior to q.

4.4 Example and Remark

We illustrate $\phi_{\mathcal{B},q}$ on the simplified example of Table 1. We consider the execution on the bottom table at the third iteration. At this stage, $\mathcal{B}$ equals $\{q_1, q_2\}$ where $q_1 = \{L_1 \to A, L_2 \to A, L_3 \to A\}$ and $q_2 = \{L_1 \to B, L_2 \to B, L_3 \to B\}$. We assume that the counter-example q is $\{L_1 \to C, L_2 \to A, L_3 \to A\}$. In this example, each context is defined precisely as each location: $c_1 = \{L_1\}$, $c_2 = \{L_2\}$, and $c_3 = \{L_3\}$. We notice that q exhibits one new tag $t = \{L_1 \to C\}$ associated with context c_1. The formula $\pi_{\mathcal{B},q}$ is then defined as $\phi_I \wedge \phi_{\succeq_\mathcal{B}} \wedge \phi_{\not\preceq_\mathcal{B}}$ where

$$\phi_{\succeq_\mathcal{B}} = L_1^{0,C}$$

and

$$\phi_{\not\preceq_\mathcal{B}} = \neg\left(L_2^{0,A} \vee L_2^{0,B}\right) \vee \neg\left(L_3^{0,A} \vee L_3^{0,B}\right).$$

A result to $\phi_{\mathcal{B},q}$ will assign C to L_1, and a value different from A and B to either L_2 or L_3 (or both). A counter-example that satisfies this formula will be superior q.

Let us discuss how ϕ_π and $\phi_{\mathcal{B},q}$ differ, which should impact the expected complexity of the SAT problem. $\phi_{\mathcal{B},q}$ is only defined on variables $v^{i,\ell}$ where $i = 0$, which should make this problem simpler. Furthermore, in $\phi_{\mathcal{B},q}$ the new tags of q are enforced, which reduce further the number of free variables. On the other hand, the number of solutions to $\phi_{\mathcal{B},q}$ is much smaller than that of ϕ_π since we are looking for much more specific counter-examples.

5 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 computational structure of the problem. Notions as tags and width have shown a useful characterisation

		Coverage			Plan Quality			Planning Time		
Domain	Classes	gCPCES	SUPERB	T1	gCPCES	SUPERB	T1	gCPCES	SUPERB	T1
LOOKANDGRAB (18)	(6,0,18,0)	**18**	**18**	15	41.87	41.87	33.73	21.59	36.46	117.49
DISPOSE (11)	(4,7,0,0)	4	6	**8**	183.50	184.00	211.75	579.58	258.94	6.00
BLOCKWORLD (3)	(0,0,3,0)	**3**	**3**	2	12.50	12.50	13.00	0.67	0.75	0.24
UTS (15)	(15,0,0,0)	**13**	**13**	11	36.40	36.40	40.60	2.73	4.29	0.23
RAOSKEYS (2)	(0,0,3,0)	**2**	**2**	1	16.00	16.00	21.00	0.59	1.20	0.45
DISPOSE-ONE (10)	(5,0,5,0)	**5**	**5**	4	61.75	68.00	79.25	29.59	67.13	376.75
WALLGRID (18)	(0,0,18,0)	**18**	**18**	4	17.50	17.50	17.50	0.73	0.86	0.09
EMPTYGRID (4)	(0,0,18,0)	**4**	**4**	**4**	17.50	17.50	17.50	0.64	1.30	0.07
BOMB (9)	(0,9,0,0)	7	**9**	**9**	106.00	106.00	101.14	95.52	3.67	0.10
COINS (9)	(0,9,0,0)	8	8	**9**	88.00	86.13	148.63	3.42	3.47	0.62

Table 2: Overview of the results among competing planners along the three main dimensions: coverage, plan length and planning time. These two last parameters are only evaluated over instances solved by all systems. Bold for best coverage. In parenthesis, number of instance. The classes column indicates with a tuple the number of instances both vertical and horizontal, non-horizontal but vertical, horizontal but non-vertical, and non-vertical non-horizontal.

of such decompositions, and have led to powerful reductions to classical planning. These decompositions have been studied extensively by Palacios et al. (2009) and Albore et al. (2010). In particular, in a close but successive work (Albore, Ramírez, 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. In this work we also use tags as a way to capture the structure of the problem; our novelty stems from showing how to do that into a non-heuristic search framework (i.e., gCPCES). Next section shows when and why this is practically beneficial, and when not.

Other approaches use a sampling based mechanism as we do, but in ways that depart substantially from how our solver, SUPERB, works. 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 $P(s_0)$. As for the sampling strategies in the CPCES framework, the sampling adopted in SUPERB explores only those states that are able to contradict some plan solution; this contrasts the work by Nguyen et al. (2012) in that SUPERB does not explore several classical planning solutions for the same initial state but iterates over an increasing set of initial states. The smarter selection of counter-examples provided by SUPERB can be seen as a more systematic way of selecting initial worlds in Kurien, Nayak, and Smith (2002) terms. Note that, SUPERB inherits the gCPCES properties, i.e, soundness and completeness in a number of iterations that is linearly bounded by the number of tags (Grastien and Scala 2018); the other strategies, i.e., refined and heuristic, can iterate a number of times that is exponential in the number of tags.

6 Experiments

We implemented our planner, SUPERB, on top of the readily available planner gCPCES by replacing the method `compute-counter-example` with `compute-optimal-counter-example` (Section 4).[1] In particular we used the same underlying classical planner FF (Hoffmann and Nebel 2001) and the SAT solver Z3 (de Moura and Bjørner 2008).

6.1 Expectations

We start with a description of how we expect our variation of gCPCES will affect the performance of the planner. We base our expectation on the structure of the problem instance, which we classify through a geometric analogy. While *width* usually refers to the size of the largest context of a problem instance, we call *height* its number of (superset maximal) contexts. We then say that an instance is *horizontal* if its height is one, and *vertical* if its width is one. This classification leads us to four overlapping classes of problem instances.

For non-horizontal instances (so, including vertical ones) we expect SUPERB to outperform gCPCES because it should be able to cover more tags in fewer iterations than gCPCES. Horizontal instances have only one tag, so a counter-example cannot be improved by the SUPERB procedure. Hence, because of its overhead, we expect SUPERB to perform slightly worse than gCPCES. Finally, non-vertical instances (so, including horizontal ones) are the hardest for planners using a mechanism based on low width (Albore, Ramírez, and Geffner 2011; Palacios and Geffner 2009). In non-vertical instances gCPCES already provides good performances w.r.t. the state of the art. We expect SUPERB

[1]The source code and the benchmarks are available at this address: bitbucket.org/enricode/cpces/.

		Planning Time		Iterations		Sampling Time		
Dom	Pro	gCPCES	SUPERB	gCPCES	SUPERB	gCPCES	SUPERB	T1 Time
MAWALLGRID	4_4_2	1.43	**1.17**	10	**7**	0.41	0.42	*0.1*
	4_4_3	20.02	**10.34**	19	**11**	0.86	1.09	*0.3*
	6_6_2	4.29	4.25	13	**12**	0.7	1.14	*0.1*
	6_6_3	1037.74	**904.75**	14	14	1.08	1.74	*4.9*
	8_8_2	124.14	**77.75**	29	**25**	2.74	3.31	*TO*
	8_8_3	TO	TO	NA	NA	NA	NA	*TO*
	10_10_2	**874.49**	1876.62	**40**	50	4.11	9.75	*TO*
	10_10_3	TO	TO	NA	NA	NA	NA	*TO*
	11_11_2	2287.07	**1606.3**	43	**38**	6.09	9.3	*TO*
	11_11_3	TO	TO	NA	NA	NA	NA	*TO*
DISPOSE	4_1	2.33	2.34	17	**17**	1.01	1.24	*0.09*
	4_2	5.07	**3.78**	32	**17**	2.05	2.59	*0.18*
	4_3	9.44	**4.99**	38	**17**	3.62	3.36	*0.39*
	4_4	13.68	**5.96**	37	**17**	5.02	4.14	*0.47*
	4_5	14.48	**8.05**	37	**17**	5.08	5.85	*0.93*
	4_6	26.15	**11.03**	38	**17**	5.58	7.07	*0.88*

Table 3: In-depth analysis on instances of MAWALLGRID, and on instances of DISPOSE. Problem cells represent grid parameters ($x \times y$) and number of agents for MAWALLGRID. For DISPOSE instance number 4 (grid size 4×4) is used for 1 to 6 objects. T1's time only reported as a reference

to improve on even further when instances are both non-vertical and non-horizontal.

6.2 Experimental Setup

To assess the usefulness of SUPERB we ran experiments over the set of benchmarks from Grastien and Scala (2018). We then compared SUPERB to gCPCES to analyse the improvement of the novel counter-example finding technique, and to T1 (Albore, Ramírez, and Geffner 2011) to evaluate how close SUPERB can get to a heuristic search engine that is notoriously very good at exploiting the structure of the problem. Depending on the competing planner, our analysis measures the amount of time needed to find a solution, the time devoted to finding the counter-examples, the number of iterations, and the quality (length) of the resulting plans.

Our benchmark suite contains 10 domains: DISPOSE, DISPOSE-ONE, BLOCKWORLD, LOOKANDGRAB, RAOSKEY, WALLGRID, EMPTYGRID, COINS, BOMB and UTS. Among these domains, DISPOSE, COINS and BOMB feature some non-horizontal and vertical instances; all other domain instances are horizontal. The following domain include non-vertical instances: BLOCKWORLD, RAOSKEY, EMPTYGRID, WALLGRID. DISPOSE-ONE. As we expect SUPERB to provide a substantial boost on non-horizontal instances, we give more attention to some of them. We also define a new domain, called MAWALLGRID, a multi-agent version of WALLGRID; the initial belief does not exclude any initial position for any agent; the goal is to get all agents in a given cell. Agents can share the same location, and similarly to WALLGRID none of them can traverse the sink position. As an additional feature, agents need to move in turn; that is, before starting to move agent i needs to wait agent $i-1$ has reached its goal position. These instances are both non-horizontal and non-vertical. Experiments were run on Ubuntu with 16GB memory on a 3.6GHz CPU. Timeout was set to 3600 secs.

	Planning Time		Iterations		Sampling Time	
Dom	gCPCES	SUPERB	gCPCES	SUPERB	gCPCES	SUPERB
LOOKANDGRAB	**27.78**	47.33	10.61	10.61	2.17	3.70
DISPOSE	579.58	**258.94**	53.75	**29.00**	58.22	83.75
BLOCKWORLD	120.04	120.05	27.75	27.75	2.66	3.48
UTS	**15.08**	27.88	16.38	16.38	4.14	8.04
RAOSKEY	**1.86**	2.49	14.00	**11.00**	0.78	1.32
DISPOSE-ONE	**29.59**	67.13	**20.50**	28.00	1.65	3.12
WALLGRID	**38.11**	55.52	33.22	33.22	3.30	5.83
EMPTYGRID	**0.64**	1.30	6.00	6.00	0.19	0.52
BOMB	95.52	**3.67**	61.00	**3.00**	19.27	2.94
COINS	3.42	3.47	12.00	**8.50**	1.55	2.14

Table 4: Instances solved by gCPCES and SUPERB.

6.3 Results and Analysis

Figure 2 provides an overview of the performance of gCPCES, SUPERB and T1. As can be observed from this high level overview, SUPERB mitigates gCPCES's inability to exploit the structure of the problem. Indeed, all the domains involving vertical instances bring SUPERB much closer to the performances of T1, whilst the benefit of gCPCES over non-vertical ones is maintained. This is particularly evident in the BOMB domain where coverage is increased by two units and speed-up is almost two orders of magnitude. In DISPOSE also, coverage is increased and runtime is improved substantially. In DISPOSE, in particular, the number of contexts grows with the number of objects to be disposed. As shown by Table 3, SUPERB exploits this decomposition in a way that the number of iterations remains constant when the number of objects grows. Comparatively, gCPCES's performance is affected in a more pronounced way. Note also the T1 remains the state of the art in DISPOSE as this domain exhibits a width equals to one. The experiments also show that SUPERB does not sacrifice plan quality for these runtime improvements.

Table 3 reports some instances for MAWALLGRID; we focus our attention on two and three agents (therefore two and three contexts) for various grid sizes. Note that T1 only manages to solve instances of the setting 4×4 and 6×6. This confirms our expectations on the performance of SUPERB in domains where the width is non-trivial and the problems are yet decomposable (non-vertical non-horizontal).

Finally, Table 4 reports an overall picture of the number of iterations, and sampling time spent by gCPCES and SUPERB. This data confirms our hypothesis that decreasing the number of iterations translates in better runtime, and also that the overhead spent by SUPERB is beneficial, especially in non-horizontal problems. Notice that in practice the improvement of counter-examples should be turned off after it has been determined that the problem instance features only one context; we did not implement this trick here in order to better understand the performance of SUPERB.

7 Conclusion

In this article we argue that a counter-example based procedure such as gCPCES for conformant planning can benefit from finding better examples. We show that one such criteria is to maximise the number of tags covered by the set

of counter-examples. This gives theoretical guarantees and proves practically beneficial too. We present SUPERB, an implementation of this idea for gCPCES, and demonstrate that it performs better for non-horizontal problems, i.e., problem instances that can be decomposed. In the "Explainable Planning" context (Fox, Long, and Magazzeni 2017), the counter-examples computed by gCPCES can be used as a justification for the resulting plan; our approach ends up with an even more compact (and thus more comprehensible) set.

This paper provides a way to combine a method that does not explicitly consider any problem structure (such as gCPCES) with techniques that do (such as the procedure for computing superior counter-examples), and to get the best of both worlds. It would be interesting to see how much further this work can be pushed and determine, for instance, which tags are redundant and can be safely ignored by the conformant planner.

Acknowledgments

Alban Grastien is with the ANU Grand Challenge *Humanising Machine Intelligence*; his work was partly performed while working for Data61.

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)*.

Albore, A.; Ramírez, 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)*.

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

Cimatti, A., and Roveri, M. 2000. Conformant planning via symbolic model checking. *J. Artif. Intell. Res.* 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.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *arXiv preprint arXiv:1709.10256*.

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)*, 97–105.

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 Brafman, R. 2006. Conformant planning via heuristic forward search: a new approach. *Artificial Intelligence (AIJ)* 170:507–541.

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.

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

Nguyen, K.; Tran, V.; Son, T.; 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)*, 190–198.

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.

Smith, D., and Weld, D. 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)*, 2033–2038.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3):90–98.