Improvements to the Generate-and-Complete Approach to Conformant Planning

Liangda Fang[1,3], **Min Zhan**[1], **Jin Tong**[1], **Xiujie Huang**[1*], **Ziliang Chen**[2], **Quanlong Guan**[1*]
[1]Jinan University, Guangzhou, 510632, China
[2]Pengcheng Laboratory, Shenzhen, 518055, China
[3]Pazhou Laboratory, Guangzhou, 510330, China
{fangld, t_xiujie, gql}@jnu.edu.cn, {zhanmin, tj16164}@stu.jnu.edu.cn, c.ziliang@yahoo.com

Abstract

Conformant planning is a computationally challenging task that generates an action sequence to achieve goal condition with uncertain initial states and non-deterministic actions. The generate-and-complete (in short, GC) approach shows superior performance on conformant planning, which iteratively enumerates the solution of a planning subproblem for a single initial state and attempts to extend it for all initial states until a conform solution is found. However, two major drawbacks of the GC approach hinder its performance: the computational overhead due to state exploration and the insertion of many redundant actions. To overcome the above drawbacks, we improve both verification and completion procedures. Experimental results show that the improved GC planner has significant improvements over the original GC approach in many instances with a large number of initial states. Our approach also outperforms all of state-of-the-art planners, solving 989 instances in comparison to 784, which is the most solved by DNF.

1 Introduction

Conformant planning [Smith and Weld, 1998] aims to generate an action sequence that guarantees goal achievement under uncertainty in initial states and non-determinism of action effects. Conformant planning is computationally challenging, more precisely, EXPSPACE-complete [Haslum and Jonsson, 1999]. Numerous approaches have been developed to tackle the inherent complexity of conformant planning.

A series of competitive approaches are based on heuristic search over belief states. Forward search-based planners: GPT [Bonet and Geffner, 2000], CMBP [Cimatti and Roveri, 2000], KACMBP [Cimatti and Roveri, 2004], conformant-FF [Hoffmann and Brafman, 2006], POND [Bryce *et al.*, 2006], T1 [Albore *et al.*, 2011], CPA(H) [Tran *et al.*, 2013], DNF, CNF, PIP [To *et al.*, 2015] start from the initial belief state and repeatedly performing executable actions to reach new belief states towards the goal condition. Backward search-based planners: CGP [Smith and Weld, 1998], YKA [Rintanen, 2002] and CAltAlt [Bryce *et al.*, 2006] work as forward search-based ones but in an opposite direction from the goal condition to the initial belief state.

Another series of competitive approaches take advantage of classical planning. Palacios and Geffner [2009] proposed a sound translation from conformant planning problems to classical planning problems. The main idea is to extend each proposition p to propositions Kp/t and $K\neg p/t$ for a set T of tags where each tag $t \in T$ is a set of unknown literals in the initial formula. Intuitively, Kp/t (resp. $K\neg p/t$) means that p (resp. $\neg p$) holds in the initial state consistent with t. When T covers the initial formula, the translation becomes complete, but leads to an exponential blowup in the size of T to the width of the problem. T0 [Palacios and Geffner, 2009] firstly attempts to obtain a solution by using a sound and polynomial but incomplete translation and then generates a solution by using a complete translation when the attempt fails.

Grastien and Scala [2020] adopted the counterexample-guided framework, originally proposed in the field of computational learning theory [Angluin, 1987] and used in address the state explosion issue in model checking [Clarke *et al.*, 2003], and developed a novel approach to conformant planning, namely CPCES. It maintains a subset S of initial sets, and computes a candidate solution α that works only for S not the whole set of initial sets. A conformant planning subproblem for S is translated to a classical planning problem via augmenting each proposition p to many copies p_s for every state $s \in S$. Unlike the translation proposed by Palacios and Geffner [2009], this translation is complete but unsound. Since the subset S is often small, a conform plan α can be computed by a classical planner. If α is incorrect, then an initial state s, in which α does not hold, is added to S. A new candidate solution, which is guaranteed to be valid for the new set S, is generated. The procedure continues until either a correct plan is found or the problem is proved unsolvable.

Nguyen *et al.* [2012] designed a conformant planner following the generate-and-complete framework (in short, GC). It firstly finds a candidate plan α that works in an initial state s and then attempts to complete α to be hold in all initial states. If the completion procedure succeeds, it finds a conformant solution for all initial states; otherwise, it tries on another plan that works in s. Both generation and completion procedures are accomplished by a classical planner. Although GC exhibits its exceptional performance

*Both are corresponding authors.

on conformant planning, it still has the following two drawbacks: (1) Several procedures of the GC approach explores all initial states. When the number of initial states is large, state exploration incurs a substantial computational overhead, making the problem difficult or even impossible to be solved within a reasonable time. (2) The completion procedure inserts many redundant actions into the solution, hence degrading the solution quality and the overall performance.

This paper is intended to overcome the above two drawbacks. The main contributions are the following. (1) We reduce the correct verification of the candidate solution to a SAT problem, which does not rely on state exploration. (2) We integrate the counterexample-guided framework into the completion procedure so as to avoid state exploration. Additionally, we devise a method to eliminate redundant segments of the candidate solution α by checking if the belief state remains unchanged after executing some segments of α. (3) We modify the main algorithm of the GC approach to accommodate the above two improvements. (4) We evaluate our improved GC approach on benchmarks that includes a total of 50 domains, with 1,179 instances from previous literature and the IPC competitions. The experimental results demonstrate that the improved GC planner outperforms the original one in two aspects: the number of solved instances and the length of solutions. Furthermore, the improved GC planner significantly improves upon every current state-of-the-art planners, solving 989 instances in comparison to 784, which is the most solved by a state-of-the-art planner DNF.

2 Conformant Planning

Throughout this paper, we fix a set $\mathcal{X}$ of propositions. We use lower case letter (*e.g.*, x, y, z) for propositions and upper case letters (*e.g.*, $\mathcal{Y}$, $\mathcal{Z}$) for subsets of propositions. A *literal* l is a proposition p or it negation $\neg p$. A *propositional formula* ϕ is built from the set $\mathcal{X}$ of propositions, the connectives $\neg, \vee, \wedge, \texttt{oneof}$ and two Boolean constants $\top$ (truth) and $\bot$ (falsity). The `oneof` connective is widely used in formulating conformant planning problems [To *et al.*, 2015; Nguyen *et al.*, 2011; Nguyen *et al.*, 2012]. Intuitively, $\texttt{oneof}(\phi_1, \cdots, \phi_n)$ means that there is an unique formula ϕ_i holds. A *term* is a conjunction of literals. For simplicity, we sometimes use a set $\{l_1, \cdots, l_n\}$ of literals to denote a term $l_1 \wedge \cdots \wedge l_n$. A *disjunctive normal form* (in short, DNF) formula is of the form $\psi_1 \vee \cdots \vee \psi_n$ and a *oneof normal form* (in short, ONF) formula is of the form $\texttt{oneof}(\psi_1, \cdots, \psi_n)$ where each ψ_i is a term.

A *state* s is a subset of propositions. Given a state s and a formula ϕ, the notation $s \models \phi$ means that s *satisfies* ϕ, which is defined as usual. A *model* of ϕ is a state satisfying ϕ. We use $[\![\phi]\!]$ for the set of models of ϕ. A *belief state* S is a set of states and can be represented as a propositional formula.

Definition 1. A deterministic conformant planning problem $\mathcal{P}$ is a tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ where

- $\mathcal{X}$: a finite set of propositions;
- $\mathcal{A}$: a finite set of actions;
- $\mathcal{I}$: a formula denoting the initial belief state;
- $\mathcal{G}$: a formula denoting the goal condition.

We say a state s is an *initial state*, iff $s \in [\![\mathcal{I}]\!]$. We define $\mathcal{P}(s) = \langle \mathcal{X}, \mathcal{A}, s, \mathcal{G} \rangle$ as a subproblem of $\mathcal{P}$ where $s \in [\![\mathcal{I}]\!]$. Definition 1 allows the initial formula $\mathcal{I}$ and the goal formula $\mathcal{G}$ to be arbitrary propositional formulas. As in [To *et al.*, 2015; Nguyen *et al.*, 2011; Nguyen *et al.*, 2012], we impose restrictions on $\mathcal{I}$ and $\mathcal{G}$: (1) $\mathcal{I}$ is restricted to a conjunction of $\mathcal{I}_l$, $\mathcal{I}_o$ and $\mathcal{I}_d$ where $\mathcal{I}_l$ is a set of literals, $\mathcal{I}_o$ is a set of ONF-formulas and $\mathcal{I}_d$ is a set of DNF-formulas, and (2) $\mathcal{G}$ is restricted to a conjunction of $\mathcal{G}_l$ and $\mathcal{G}_d$ where $\mathcal{G}_l$ is a set of literals and $\mathcal{G}_d$ is a set of DNF-formulas.

Each action $a \in \mathcal{A}$ is defined as a pair $\langle \texttt{pre}(a), \texttt{eff}(a) \rangle$ where $\texttt{pre}(a)$ is a set of literals indicating the precondition and $\texttt{eff}(a)$ is a non-empty set of conditional effects representing the effect. A conditional effect e is defined as a pair $\langle \texttt{con}(e), \texttt{lit}(e) \rangle$ where $\texttt{con}(e)$ and $\texttt{lit}(e)$ are sets of literals. Intuitively, if every literal of $\texttt{con}(e)$ holds before the execution of e, then every literal of $\texttt{lit}(e)$ holds after; otherwise, nothing is changed. A conditional effect e is *applicable* in s, iff $s \models \texttt{con}(e)$. The *positive condition* $\texttt{con}^+(x, a)$ of an action a for a proposition x is defined as $\bigvee_{e \in \texttt{eff}(a), x \in \texttt{lit}(e)} \texttt{con}(e)$. Similarly, the *negative condition* $\texttt{con}^-(x, a)$ is $\bigvee_{e \in \texttt{eff}(a), \neg x \in \texttt{lit}(e)} \texttt{con}(e)$. Intuitively, the positive (resp. negative) condition holds before performing a implies that x become true (resp. false) after. The *positive effect* $\texttt{eff}^+(s, a)$ of an action a in a state s is a set of propositions defined as $\{x \mid e \in \texttt{eff}(a), s \models \texttt{con}(e), x \in \texttt{lit}(e)\}$. Similarly, the *negative effect* $\texttt{eff}^-(s, a)$ is defined as $\{x \mid e \in \texttt{eff}(a), s \models \texttt{con}(e), \neg x \in \texttt{lit}(e)\}$. We require every action a to be *self-contradictory*, formally, $\texttt{eff}^+(s, a) \cap \texttt{eff}^-(s, a) = \emptyset$ for every state s.

The resulting state $\tau(s, a)$ of executing an action a in a state s is $s \setminus \texttt{eff}^-(s, a) \cup \texttt{eff}^+(s, a)$. The resulting state $\Gamma(s, [a_1, \cdots, a_n])$ of executing an action sequence $[a_1, \cdots, a_n]$ in a state s is recursively defined by $\Gamma(s, [a_1, \cdots, a_{n-1}, a_n]) = \tau(\Gamma(s, [a_1, \cdots, a_{n-1}]), a_n)$ and $\Gamma(s, \varepsilon) = s$ where ε is an empty sequence. An action a is *executable* in a state s iff $s \models \texttt{pre}(a)$. An action sequence $[a_1, \cdots, a_n]$ is *executable* in a state s iff $s \models \texttt{pre}(a_1)$ and $\Gamma(s, [a_1, \cdots, a_i]) \models \texttt{pre}(a_{i+1})$ for $i \geq 1$.

Given an action sequence $\alpha : [\alpha_1, \cdots, \alpha_n]$, we use α_i for the i-th action a_i. An action sequence α is a *solution* to $\mathcal{P}$, iff it is executable in every initial state and every resulting state by performing α from every initial state is a goal state.

We illustrate conformant planning with the *robot navigation* problem proposed in [Cimatti and Roveri, 2004].

Example 1. *A robot can move in four directions in a* 5×5 *grid. Each cell* (i, j) *in the grid can be represented by two propositions* x^i *and* y^j*. For example,* $x^1 \wedge y^2$ *means that the robot is at cell* $(1, 2)$*. The robot initially start at any position and its goal is to reach the center cell* $(3, 3)$*.*

- $\mathcal{X} = \{x^i \mid 1 \leq i \leq 5\} \cup \{y_j \mid 1 \leq j \leq 5\}$;
- $\mathcal{I} = \texttt{oneof}(x^1, \cdots, x^5) \wedge \texttt{oneof}(y^1, \cdots, y^5)$;
- $\mathcal{G} = x^3 \wedge y^3$;
- $\mathcal{A} = \{\texttt{left}, \texttt{right}, \texttt{up}, \texttt{down}\}$;
- $\texttt{pre}(\texttt{left}) = \texttt{pre}(\texttt{right}) = \texttt{pre}(\texttt{up}) = \texttt{pre}(\texttt{down}) = \emptyset$;

Algorithm 1: GC($\mathcal{P}$)

Input: $\mathcal{P} = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$: a conformant planning problem
Output: α: a solution to $\mathcal{P}$

1 $\mathcal{I}' \leftarrow \texttt{Combine}(\mathcal{I})$
2 $s \leftarrow$ a state of $[\![\mathcal{I}']\!]$
3 **repeat**
4 $\alpha \leftarrow \texttt{FindNextPlan}(\mathcal{P}(s))$
5 **if** $\alpha = \emptyset$ **then return** $\emptyset$;
6 $\alpha \leftarrow \texttt{Complete}(\mathcal{P}, [\![\mathcal{I}']\!], \alpha)$
7 **if** $\texttt{Verify}(\mathcal{P}, \alpha) \neq \emptyset$ **then return** α ;
8 **until true**;

Algorithm 2: Complete($\mathcal{P}, S, \alpha$)

Input: $\mathcal{P} = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$: a conformant planning problem;
S: a set of initial states;
α: a plan to be completed.
Output: α: a possible solution to $\mathcal{P}$.

1 **for** $i = 1$ **to** k **do**
2 **foreach** $s \in S$ **do**
3 **if** $\texttt{VerifyClassicPlan}(\mathcal{P}(s), \alpha) =$ **true then**
4 $s' \leftarrow s$
5 **continue**
6 $\alpha \leftarrow \texttt{Extend}(\mathcal{P}, s, s', \alpha)$
7 **if** $\alpha = \emptyset$ **then return** $\emptyset$;
8 **return** α

- $\texttt{eff}(\texttt{left}) = \{\langle \{x^{i+1}\}, \{x^i, \neg x^{i+1}\} \rangle \mid 1 \leq i \leq 4\}$;
- $\texttt{eff}(\texttt{right}) = \{\langle \{x^i\}, \{x^{i+1}, \neg x^i\} \rangle \mid 1 \leq i \leq 4\}$;
- $\texttt{eff}(\texttt{up}) = \{\langle \{y^{i+1}\}, \{y^i, \neg y^{i+1}\} \rangle \mid 1 \leq i \leq 4\}$;
- $\texttt{eff}(\texttt{down}) = \{\langle \{y^i\}, \{y^{i+1}, \neg y^i\} \rangle \mid 1 \leq i \leq 4\}$.

One solution to this problem is $[\texttt{left} \times 4, \texttt{up} \times 4, \texttt{right} \times 2, \texttt{down} \times 2]$. *The plan α is always executable since all actions have valid precondition. She can move from any initial cell to the upper-left corner (x^1, y^1) via performing four consecutive* up *actions followed by four consecutive* left *actions. She finally reaches the center cell (x^3, y^3) by executing two consecutive* right *and two consecutive* down *actions.* □

3 The Generate-And-Complete Approach

In this section, we briefly introduce the *generate-and-complete* approach to conformant planning, proposed by Nguyen *et al.* [2012].

The GC approach utilizes a classical planner Ω, which is required to not only generate one single solution for a classical planning problem but also enumerate many different solutions in an iterative manner. The procedure FindNextPlan aims to iteratively generate a new solution via utilizing a classical planner Ω and it returns $\emptyset$ when Ω cannot find any new solution. The basic idea behind the GC approach, illustrated in Algorithm 1[1], is as follows. It first generates a solution α of the subproblem $\mathcal{P}(s)$ where s is an initial state if it exists (line 4). Since α may not be a solution of the original problem $\mathcal{P}$, it then attempts to repair α to account for all of initial states (line 6). If the completion procedure fails to repair α, then it continues to find the next solution of $\mathcal{P}(s)$. The above steps repeat until no more solution of $\mathcal{P}(s)$ is found by Ω (line 5) or a solution of $\mathcal{P}$ is generated (line 7).

[1]The pseudocodes of Algorithms 1 and 2 are slightly different from that in [Nguyen *et al.*, 2012] since the pseudocodes in our paper is based on the latest implementation of the GC[LAMA] planner.

Algorithm 3: Extend($\mathcal{P}, s, s', \alpha$)

Input: $\mathcal{P} = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$: a conformant planning problem;
s: the current state;
s': the state where α is a solution to $\mathcal{P}(s')$;
α: a plan to be extended.
Output: α: a solution to $\mathcal{P}(s)$.

1 $s_0 \leftarrow s$
2 **for** $j = 1$ **to** $|\alpha|$ **do**
3 $\mathcal{G}' \leftarrow \texttt{GenerateGoal}(\alpha_j, \Gamma(s', [\alpha_1, \cdots, \alpha_{j-1}]))$
4 $\beta^j \leftarrow \texttt{FindNextPlan}(\langle \mathcal{X}, \mathcal{A}, s_{j-1}, \mathcal{G}' \rangle)$
5 **if** $\beta^j = \emptyset$ **and** $s_{j-1} \not\models \texttt{pre}(a_j)$ **then return** $\emptyset$;
6 $s_j \leftarrow \Gamma(s_{j-1}, \beta^j \circ \alpha_j)$
7 $\beta^{\mathcal{G}} \leftarrow \texttt{FindNextPlan}(\mathcal{P}(s_{|\alpha|}))$
8 **if** $\beta^{\mathcal{G}} = \emptyset$ **then return** $\emptyset$;
9 $\alpha \leftarrow \beta^1 \circ \alpha_1 \circ \cdots \circ \beta^{|\alpha|} \circ \alpha_{|\alpha|} \circ \beta^{\mathcal{G}}$
10 **return** α

We say a conformant planner is *sound* iff it returns a plan α only if α is a solution to the conformant planning problem $\mathcal{P}$; it is *complete* iff it returns $\emptyset$ only if $\mathcal{P}$ has no solution; and it is *strongly complete* for k iff it enumerates all solutions to $\mathcal{P}$ with length up to k, and returns $\emptyset$ when such all solutions exhaust. The above properties for classical planner can be similarly defined. Since the correctness of α is guaranteed in line 7, the GC approach is sound. Haslum and Jonsson [1999] proved that if a conformant planning problem has a solution, then it has a solution with length up to $2^{2^{|\mathcal{X}|}}$ where $|\mathcal{X}|$ is the number of propositions. In addition, every solution to $\mathcal{P}$ is also a solution to $\mathcal{P}(s)$. In summary, if the underlying classical planner Ω is strongly complete for $2^{2^{|\mathcal{X}|}}$, then the GC approach is complete. The completeness of Ω cannot ensure the completeness of GC since Ω is not guaranteed to generate all solutions to $\mathcal{P}(s)$ with length up to $2^{2^{|\mathcal{X}|}}$ and the completion procedure is incomplete.

There are three remaining issues of Algorithm 1: (1) how to obtain an initial state s; (2) how to verify the correctness of a candidate plan α; and (3) how to repair a candidate plan α.

For the first issue, Algorithm 1 enumerates a collection $[\![\mathcal{I}]\!]$ of initial states and choose one state from $[\![\mathcal{I}]\!]$ as the initial state s (lines 1 - 2). In general, large-scale planning problems involves an enormous number of initial states. To reduce the number of initial states, Tran *et al.* [2013] proposed oneof-*combination* technique that generates a new initial formula $\mathcal{I}'$ by combining some ONF-formulas of the initial formula $\mathcal{I}$ into one. The formula $\mathcal{I}'$ has two characteristics: (1) $\mathcal{I}'$ entails $\mathcal{I}$; and (2) every solution of the new conformant planning problem $\langle \mathcal{X}, \mathcal{A}, \mathcal{I}', \mathcal{G} \rangle$ is also a solution of the original problem $\mathcal{P}$ and vice versa. For the details of oneof-combination technique, please refer to [Tran *et al.*, 2013].

The verification procedure addresses the second issue. The

GC approach reduces the problem of checking if α is a solution to every classical planning problem $\mathcal{P}(s)$ for every state $s \in [\![\mathcal{I}']\!]$. The latter can be solved in the following. It applies the effects of actions $\alpha_1, \cdots, \alpha_i$ progressively to obtains each immediate state $\Gamma(s, [\alpha_1, \cdots, \alpha_i])$ for $0 \leq i \leq |\alpha|$. The plan α is a solution to $\mathcal{P}(s)$ iff (1) α is executable in s; and (2) the final state $\Gamma(s, \alpha)$ satisfies the goal condition $\mathcal{G}$.

The completion procedure, illustrated in Algorithm 2, aims to fix the third issue. It attempts to extend the solution α of the subproblem $\mathcal{P}(s)$ to a solution to $\mathcal{P}$. For each initial state $s \in S$, the inner loop (lines 2 - 7) sequentially inserts a few actions into α. At each iteration, s' is the last state where no action is needed to be inserted (line 4). Based on s' and the current initial state s, actions are inserted into α to ensure that α becomes a solution for $\mathcal{P}(s)$ (line 6), if this is impossible, the algorithm terminates (line 7). The completed sequence α is not guaranteed to remain a solution to the subproblem for previously visited states. As a result, the inner loop will be invoked by k times where $k = 3$ in the implementation.

Algorithm 3 inserts some actions into the solution α to a subproblem $\mathcal{P}(s')$ so that α becomes a solution to $\mathcal{P}(s)$. An action sequence β^j is first generated by using the underlying classical planner and then is inserted between α_{j-1} and α_j (line 4). To this end, an immediate goal $\mathcal{G}'$ is generated based on action α_j and state $s'_j : \Gamma(s', [\alpha_1, \cdots, \alpha_{j-1}])$ (line 3). Three strategies for generating $\mathcal{G}'$ are proposed in [Nguyen *et al.*, 2012]: *Ignorant*: $\texttt{pre}(\alpha_j)$; *Greedy*: $\texttt{pre}(\alpha_j) \wedge \bigwedge_{e \in \texttt{eff}(\alpha_j), s'_j \models \texttt{con}(e)} \texttt{con}(e))$; and *Hybrid*: $\texttt{pre}(\alpha_j) \wedge \bigwedge_{e \in \texttt{eff}(\alpha_j), s'_j \models \texttt{con}(e), |con(e)| \leq 1} \texttt{con}(e)$. All strategies require α_j to be executable. The greedy strategy ensures that the execution of α_j preserves the execution of every applicable conditional effect of α_j in s'_j while the ignorant strategy does not. The hybrid strategy is a balance between them, which considers only applicable conditional effects whose condition is $\top$ or a single literal. The empirical results reported in [Nguyen *et al.*, 2012] demonstrated that the hybrid strategy outperforms the other two. If the subproblem $\langle \mathcal{X}, \mathcal{A}, s_j, \mathcal{G}' \rangle$ has no solution and α_j is not executable in state s_j, then repairing α to fit s is impossible and the algorithm terminates without a solution (line 5). Otherwise, we compute the resulting state s_{j+1} of executing the sequence $\beta^i \circ \alpha_j$ in s_j (line 6). To achieve the goal condition $\mathcal{G}$, an additional action sequence $\beta^{\mathcal{G}}$ is generated in a similar way to β^j (lines 7 and 8). Finally, we insert each subsequence β^j between α_{j-1} and α_j and append $\beta^{\mathcal{G}}$ to α (line 9).

4 Some Improvements to the GC Approach

In this section, we first analyze two drawbacks of the GC approach (Sec. 4.1). Then, we observe that the drawbacks lie in the two verification and completion procedures and hence propose SAT-based verification procedure (Sec. 4.2) and counterexample-guided completion procedure (Sec. 4.3). Finally, we integrate the two above improved procedures into the main GC algorithm (Sec. 4.4).

4.1 Two Drawbacks of the GC Approach

Although the GC approach is effective in conformant planning, it has the following two significant drawbacks. (1) In some conformant planning problems, even using one-of combination technique, the number of initial states is still extremely large. For example, the or-dispose problem with 4 objects on 4×4 grids has 2^{48} initial states. The overall efficiency of GC drops considerably since the three procedures: state enumeration, completion and verification traverses through all initial states. Even the state enumeration procedure sometimes runs timeout. (2) The complete procedure often introduces a substantial number of redundant actions. This results in an overly long solution, which negatively impacts the performance of verification procedure and the overall efficiency of the GC approach. For example, in the uts-l problem with 90 nodes, the GC approach generates a solution with length 4269, approximately 15 times longer than the shortest solution with only length 268.

4.2 SAT-Based Verification Procedure

We adopt a SAT-based verification procedure, proposed in [Hoffmann and Brafman, 2006; Grastien and Scala, 2020], which is a reduction from correctness verification of candidate solutions to a SAT problem. The major advantage of the SAT-based method over the verification in the original GC approach is that it does not exhaustively enumerates all initial states.

Suppose that $\mathcal{P} = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ is a conformant planning problem and α is an action sequence. To distinguish the truth value of a proposition x on different time points during performing α, we use x_0 to represent the value of x in the initial state, and x_i to denote the value of x after executing the prefix $[\alpha_1, \cdots, \alpha_i]$ one by one. Given a formula ϕ and an index i, we use ϕ_i for the formula resulting from ϕ via replacing every occurrence of proposition x by the i-th time-stamped version x_i.

The core of the SAT-based verification procedure is the successor state axiom $\psi_{i,x,a}$ for proposition x and action a at time point i, which is defined as:

$$x_i \leftrightarrow [\texttt{con}^+(x, a)_{i-1} \vee (x_{i-1} \wedge \neg \texttt{con}^-(x, a)_{i-1})].$$

Intuitively, $\psi_{i,x,a}$ means that at the time point i, x holds after executing a iff the positive condition of a for x holds before, or x holds before and the negative condition does not.

The trace axiom $\phi_{\mathcal{P},\alpha}$ is defined as

$$\mathcal{I}_0 \wedge \bigwedge_{0 < i \leq |\alpha|} \left(\neg \texttt{pre}(\alpha_i)_{i-1} \wedge \bigwedge_{x \in \mathcal{X}} \psi_{i,x,\alpha_i}\right) \wedge \neg \mathcal{G}_{|\alpha|}.$$

Let ϕ be a formula over $\bigcup_{0 \leq i \leq |\alpha|} \mathcal{X}_i$. We say a state s is the *initial-projection* of an assignment t of ϕ, if for every $x \in \mathcal{X}$, we have $x \in s$ iff $x_0 \in t$.

The following theorem ensures that if the SAT-based verification procedure produces an initial state s, then α is not a solution to $\mathcal{P}(s)$.

Theorem 1 ([Grastien and Scala, 2020])**.** *A state s is the initial-projection of an assignment of $\phi_{\mathcal{P},\alpha}$ iff α is not a solution to $\mathcal{P}(s)$ and $s \in [\![\mathcal{I}]\!]$.*

As a corollary, if the SAT-based verification procedure returns no assignment, then α is a solution to $\mathcal{P}$, and vice versa.

Corollary 1 ([Grastien and Scala, 2020])**.** *The action sequence α is a solution to $\mathcal{P}$ iff $\phi_{\mathcal{P},\alpha}$ is unsatisfiable.*

Algorithm 4: CG-Complete$(\mathcal{P}, s', \alpha)$

Input: $\mathcal{P} = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$: a conformant planning problem;
s': a state;
α: a plan to be completed.
Output: α: a possible solution to $\mathcal{P}$.

1 $S \leftarrow \{s'\}$
2 **repeat**
3 $s \leftarrow \texttt{Verify}(\mathcal{P}, \alpha)$
4 **if** $s = \emptyset$ **then return** α ;
5 $S \leftarrow S \cup \{s\}$
6 $\alpha \leftarrow \texttt{Extend}(\mathcal{P}, s, s', \alpha)$
7 $s' \leftarrow s$
8 **if** $\alpha = \emptyset$ **then return** α ;
9 $\alpha \leftarrow \texttt{EliminateRedundantAct}(S, \alpha)$
10 **until true**;
11 **return** α

The trace axiom $\phi_{\mathcal{P},\alpha}$ involves $|\mathcal{X}| \times (|\alpha|+1)$ propositions where $|\mathcal{X}|$ is the number of propositions and $|\alpha|$ is the length of α. The performance of SAT solvers is highly sensitive to the number of propositions and the length of the formula. To accelerate the verification phase, we will simplify the trace axiom via forgetting some redundant propositions. Due to space limit, we introduce the simplification of trace axioms in extended version.

Example 2. *We continue the robot navigation problem illustrated in Example 1. Suppose that a plan* $\alpha = [\texttt{left}, \texttt{up}]$. *We first construct the successor state axiom. Since* left *action contains only the conditional effect* $\langle \{x^2\}, \{x^1, \neg x^2\} \rangle$ *affecting* x^1, *the positive condition* $\texttt{con}^+(x^1, \texttt{left})$ *of* left *for* x^1 *is* x^2 *and the negative condition* $\texttt{con}^-(x^1, \texttt{left})$ *is* $\top$. *Hence, the successor state axiom* $\psi_{1,x^1,\texttt{left}}$ *is* $x_1^1 \leftrightarrow x_0^2 \vee (x_0^1 \wedge \top)$. *The other successor state axioms can be similarly constructed.*

Finally, the trace axiom $\phi_{\mathcal{P},\alpha}$ *is the conjunction of the following formulas:*

- $\texttt{oneof}(x_0^1, \cdots, x_0^5) \wedge \texttt{oneof}(y_0^1, \cdots, y_0^5)$;
- $\bigwedge_{i=1}^{5} \left(\psi_{1,x_i,\texttt{left}} \wedge \psi_{1,y_i,\texttt{left}} \wedge \psi_{2,x_i,\texttt{up}} \wedge \psi_{2,y_i,\texttt{up}}\right)$;
- $\neg(x_2^3 \wedge y_2^3)$.

It can be verified that $\phi_{\mathcal{P},\alpha}$ *is satisfiable and hence* α *is not a solution to* $\mathcal{P}$. □

4.3 Counterexample-Guided Completion Procedure

We will introduce two improvements to the completion procedure, shown in Algorithm 4. The first improvement is to adopt the counterexample-guided (CG) framework to avoid explicit state explosion drawback. The CG completion procedure maintains a subset S of initial states that is used to eliminate redundant actions, which will be illustrated later. Initially, it contains only one state s' (line 1). It firsts checks if α is a solution of $\mathcal{P}$. If it fails the verification, it will generate an initial state s, which serves as a witness to the fact that α is not a solution to $\mathcal{P}$ and add s into S (lines 3 and 5). Unlike the original completion procedure, the improved one does not iterate over $[\![\mathcal{I}']\!]$ to extend the candidate plan α. Instead, it extends α only for each newly generated initial state s (line 6). Afterward, s' will be set to the witness s for the next iteration since α is extended to be a solution of $\mathcal{P}(s)$ (line 7). The above steps repeat until a solution to $\mathcal{P}$ is found (line 4) or α cannot be extend to a solution to $\mathcal{P}(s)$ (line 8).

During the extension process, several redundant actions may be introduced into the candidate plan α, which degrades both the quality of solutions and the efficiency of the verification procedure. To mitigate this drawback, we will remove redundant actions from α (line 9). We start from an initial belief state S_0, and generate an evolving sequence of belief states $[S_0, \cdots, S_n]$ by sequentially executing each action of α where $S_0 = S$ and $S_i = \{\Gamma(s, [\alpha_1, \cdots, \alpha_i]) \mid s \in S_0\}$ for $1 \leq i \leq |\alpha|$. If two belief states S_i and S_j are identical, then the action subsequences $[\alpha_{i+1}, \cdots, \alpha_j]$ can be considered redundant. This is because, for each state $s \in S$, the original candidate plan α is a solution to $\mathcal{P}(s)$ iff the removed plan $[\alpha_1, \cdots, \alpha_i, \alpha_{j+1}, \cdots, \alpha_{|\alpha|}]$ is a solution to $\mathcal{P}(s)$. We repeatedly seek for two identical belief states S_i and S_j and remove the segment $[\alpha_{i+1}, \cdots, \alpha_j]$ from α until all belief states are distinct. The experimental evaluation shows that each S_i has at most 3789 states. Hence, each S_i is explicitly represented as a set and this implementation of the elimination process is effective, which costs at most 200s.

Example 3. *Table 1 shows the run of counterexample-guided completion procedure for the navigation robot problem. Algorithm 4 takes* $s' = \{x^5, y^5\}$ *and* $\alpha = [\texttt{left}, \texttt{up}, \texttt{left}, \texttt{up}]$ *as input. At the 1st iteration, the SAT-based verification procedure prove that* α *is not a solution to the navigation robot problem with an initial state* $\{x^4, y^4\}$. *Now, the belief state* S *contains two states* $\{x^4, y^4\}$ *and* $\{x^5, y^5\}$. *An action sequence* $[\texttt{left}, \texttt{right} \times 2, \texttt{up}, \texttt{down} \times 2]$ *will be inserted inbetween the first* left *action and the first* up *action of* α *so that* α *becomes a solution for the state* $\{x^4, y^4\}$. *Algorithm 4 discovers that* $S = \{\Gamma(s, [\texttt{left}, \texttt{right}]) \mid s \in S\}$ *and that* $\{\Gamma(s, [\texttt{left}, \texttt{right} \times 2]) \mid s \in S\} = \{\Gamma(s, [\texttt{left}, \texttt{right} \times 2, \texttt{up}, \texttt{down}]) \mid s \in S\}$. *So the two segments* $[\texttt{left}, \texttt{right}]$ *and* $[\texttt{up}, \texttt{down}]$ *can be eliminated from* α. *We obtain a simplified candidate plan* $\alpha : [\texttt{right}, \texttt{down}, \texttt{left}, \texttt{up}, \texttt{left}, \texttt{up}]$.

However, the above plan α *is not the solution for every initial state. At the 2nd iteration, the verification procedure returns a witness* $\{x^1, y^1\}$ *that is added into* S. *Then,* α *is extended to an plan* $[\texttt{right} \times 3, \texttt{down} \times 3, \texttt{right}, \texttt{down}, \texttt{left}, \texttt{up}, \texttt{left}, \texttt{up}]$. *None of actions in the above plan is redundant according to the three states:* $\{x^1, y^1\}$, $\{x^4, y^4\}$ *and* $\{x^5, y^5\}$. *Finally, the new plan* α *is a solution to the navigation robot problem.* □

4.4 The Improved Main Algorithm

The last two subsections introduce improvements to the verification and completion procedures. We are ready to propose the modification to the main algorithm of the GC approach, illustrated in Algorithm 5. The improved GC approach works as the original GC approach except the following. (1) It generates a model of $\mathcal{I}'$ as the initial state s via utilizing the SAT solver rather than firstly enumerates all states of $[\![\mathcal{I}']\!]$ and then select one of $[\![\mathcal{I}']\!]$ as s (lines 1 and 2). (2) It uses SAT-based verification procedure and CG completion procedure instead the original two procedures. (3)

s	Extended Solutions	Eliminated Solutions
(x_4, y_4)	$[\texttt{left}, \texttt{right} \times 2, \texttt{up}, \texttt{down} \times 2, \texttt{left}, \texttt{up}, \texttt{left}, \texttt{up}]$	$[\texttt{right}, \texttt{down}, \texttt{left}, \texttt{up}, \texttt{left}, \texttt{up}]$
(x_1, y_1)	$[\texttt{right} \times 3, \texttt{down} \times 3, \texttt{right}, \texttt{down}, \texttt{left}, \texttt{up}, \texttt{left}, \texttt{up}]$	$[\texttt{right} \times 3, \texttt{down} \times 3, \texttt{right}, \texttt{down}, \texttt{left}, \texttt{up}, \texttt{left}, \texttt{up}]$

Table 1: A run of CG-Completion Procedure for Navigation Robot Problem.

Algorithm 5: iGC($\mathcal{P}$)

Input: $\mathcal{P} = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$: a conformant planning problem
Output: α: a solution to $\mathcal{P}$

```
1 I' ← Combine(I)
2 s ← a state of [[I']]
3 repeat
4     α ← FindNextPlan(P(s))
5     if α = ∅ then return ∅;
6     α ← CG-Complete(P, s, α)
7     if α ≠ ∅ then return α ;
8 until true;
```

The verification procedure is invoked in the CG completion procedure rather than the main algorithm.

Similarly to the GC approach, we obtain the following two theorems regarding the soundness and completeness of the improved GC approach.

Theorem 2 (Soundness)**.** *If Algorithm iGC returns a plan α for $\mathcal{P}$, then α is a solution to $\mathcal{P}$.*

Theorem 3 (Completeness)**.** *Let Ω be the underlying classical planner. If Ω is strongly complete for $2^{2^{|\mathcal{X}|}}$ where $|\mathcal{X}|$ is the number of propositions and $\mathcal{P}$ has a solution, then Algorithm iGC will terminate with a solution to $\mathcal{P}$.*

4.5 Comparisons to CPCES

We illustrate the main differences between CPCES and our approach in the following. (1) The main algorithm of our approach is still the generate-and-complete approach while CPCES is based on the counterexample-guided framework. We only integrate the CG framework into completion procedure. (2) When a candidate plan α is not a solution, CPCES completely discards α and computes a new solution. In contrast, we attempt to augment α to make it valid for all initial states.

5 Experimental Evaluation

5.1 Implementation and Experimental Setup

We implemented the improved GC approach, namely iGC, on top of the source code of the GC[LAMA] planner. Like GC[LAMA], we use LAMA [Richter and Westphal, 2010] as the underlying classical planner. Since the trace formula defined in Section 4.2 is arbitrary propositional formula, we employ Z3 [de Moura and Bjørner, 2008] as the underlying SAT solver of the verification procedure. For the generation of the immediate goal G' (line 3 in Algorithm 3), we adopt the greedy strategy, which is able to solve domains like btc, grid-empty and grid-wall, which are solved by neither ignorant nor hybrid strategies.

We collect 50 domains with a total of 1179 instances, including 7 domains from IPC-5 [Bonet and Givan, 2006]: adder-ipc5, coins, comm, uts-k, uts-l, uts-r, and sortnet; 6 domains from IPC-6 [Bryce and Buffet, 2008]: uts-cycle, raos-key, forest, dispose, adder-ipc6, and block; 7 domains from [Hoffmann and Brafman, 2006]: logistics, bomb, ring, grid, omlette, cleaner, and safe; 7 domains from [Palacios and Geffner, 2009]: cube-center, sqr-center, corners-sqr, corners-cube, look-and-grab, 1-dispose, and push-to; 10 domains from [To *et al.*, 2015]: new-dispose, new-push, new-ring, new-uts-cycle, new-uts-k, or-1-dispose, or-coins, or-dispose, or-new-push, and or-push-to; 2 domains from [Grastien and Scala, 2020]: grid-empty and grid-wall; 5 domains from [Nguyen *et al.*, 2012]: 1-dispose-disappear, hall-a, hall-r, marker-enc1, and look-and-grab-disappear; 2 domains from [Eiter *et al.*, 2003]: bt and btc; 3 domains from others: retrieve, reward and to-trash; and 1 domain we propose: or-1-dispose-disappear, a variant of or-1-dispose, which action pickup randomly drops the item at any location if it is held.

We compare our approach, namely iGC, with 8 state-of-the-art planners: DNF, CNF, PIP, T1, CPA(H), gCPCES, iCPCES and GC[LAMA]. The experiments were run on Ubuntu 24.04, with an Intel 8086K 4.0 GHz CPU and 64GB RAM. The memory limit is 16GB, and the time limit is 3600s.

5.2 Comparative Analysis

We first make comparisons among 8 planners together with the iGC in terms of coverage of solved instance. Then, we compare the iGC with three most closely solvers GC[LAMA], gCPCES and iCPCES in detail.

Table 2 summarizes the total number of instances solved by each planner in each domain. We can make several observation from Table 2. (1) iGC solves the highest number of instances (989), followed by DNF, which ranks second with 784 instances. Moreover, iGC solves the most instances in 29 domains and ranks second in 8 domains. (2) It is worth noting that iGC is the only planner capable of solving 77 instances across 6 domains: 1-dis-dis, look-and-grab, new-dispose, or-1-dis-dis, ring, to-trash. (3) iGC is able to solve all instances in 24 domains, while GC and DNF handles all instances in 20 and 16 domains, respectively. (4) iGC has the same performance as gCPCES in the challenging domains grid-empty and grid-wall introduced in [Grastien and Scala, 2020]. Other planners except T1 are unable to solve any instance of the above two domains. (5) For 101 instances with high uncertainty in the initial state (including or-1-dis-dis, or-1-dis, or-coins, or-dispose, or-new-push, and or-push-to), iGC solves a total of 56 instances, which is just fewer than 84 instances solved by CNF and 78 instances solved by PIP. In particular, in the or-1-dis-dis domain, iGC outperforms CNF and PIP by solving 11 instances as opposed to 5 solved by CNF and PIP. (6) iGC fails to solve any instance on some domains. In domains: 1-dis, or-1-dis, omlette, marker-enc1 and new-uts-k, the underlying planner

Domains	DNF	CNF	PIP	T1	CPA(H)	iCPCES	gCPCES	GC	iGC
1-dis-dis(90)	24	9	4	36	19	23	10	41	**65**
1-dis(21)	**21**	11	8	14	17	9	10	0	0
adder-ipc5(4)	**2**	0	0	0	0	ER	ER	ER	ER
adder-ipc6(4)	**1**	0	0	0	**1**	ER	ER	ER	ER
blocks(5)	3	2	2	2	4	2	4	**5**	**5**
bomb(21)	20	**21**	**21**	20	10	ER	19	**21**	**21**
bt(19)	**19**	**19**	**19**	18	**19**	**19**	**19**	**19**	**19**
btc(19)	**19**	**19**	**19**	**19**	**19**	**19**	**19**	0	**19**
cleaner(10)	**10**	9	9	**10**	**10**	**10**	9	3	3
coins(25)	16	15	15	**25**	15	15	15	15	**25**
comm(25)	**25**	15	15	**25**	**25**	ER	**25**	**25**	**25**
cor-sqr(19)	**19**	**19**	**19**	18	**19**	**19**	0	**19**	**19**
cor-cube(22)	**22**	**22**	**22**	17	20	16	0	**22**	**22**
cube-center(37)	36	24	27	**37**	36	17	19	**37**	**37**
dispose(98)	97	86	85	91	82	81	41	**98**	97
forest(8)	3	1	1	6	1	1	7	**8**	**8**
grid(15)	13	**15**	13	**15**	11	**15**	**15**	**15**	**15**
grid-empty(18)	ER	ER	ER	**4**	ER	ER	**4**	0	**4**
grid-wall(18)	ER	ER	ER	4	0	ER	**18**	0	**18**
hall-a(21)	18	9	8	11	**21**	18	3	18	18
hall-r(11)	**11**	**11**	6	**11**	**11**	1	0	**11**	**11**
logistics(17)	13	13	13	**17**	12	5	16	16	16
look-and-grab-dis(18)	2	1	7	12	9	10	0	**18**	**18**
look-and-grab(80)	71	47	61	25	65	35	3	79	**80**
marker-enc1(4)	ER	ER	ER	0	**2**	ER	ER	0	0
new-dispose(6)	ER	ER	ER	2	4	5	ER	2	**6**
new-push(12)	**12**	9	11	9	2	8	7	2	8
new-ring(11)	**11**	6	4	0	7	ER	**11**	7	9
new-uts-cycle(13)	**13**	1	1	0	0	ER	0	1	ER
new-uts-k(10)	**10**	6	5	3	5	5	0	0	0
omlette(5)	ER	ER	ER	**5**	0	ER	0	0	0
or-1-dis-dis(16)	5	5	5	0	ER	ER	5	0	**11**
or-1-dis(16)	8	**16**	11	2	ER	7	5	0	0
or-coins(10)	1	**10**	**10**	**10**	ER	4	4	0	9
or-dispose(23)	9	**23**	**23**	19	ER	18	11	0	16
or-new-push(20)	4	**19**	16	6	ER	4	5	0	12
or-push-to(16)	6	11	**13**	3	ER	3	1	0	8
push-to(108)	103	88	91	68	61	20	33	**108**	**108**
raos-key(8)	**2**	1	1	1	**2**	**2**	**2**	**2**	**2**
retrieve(15)	9	8	9	10	7	9	5	**15**	14
reward(15)	9	4	2	9	7	6	5	**15**	12
ring(100)	10	4	2	39	8	3	47	12	**100**
safe(6)	**6**	5	3	**6**	**6**	**6**	**6**	**6**	**6**
sortnet(15)	8	8	8	0	**15**	ER	ER	**15**	**15**
sqr-center(30)	**30**	26	29	**30**	**30**	21	**30**	29	**30**
to-trash(15)	3	1	0	5	6	4	0	10	**12**
uts-cycle(28)	10	1	1	0	ER	ER	0	**15**	14
uts-k(19)	17	5	2	13	14	12	14	**19**	**19**
uts-l(17)	**17**	5	2	**17**	12	**17**	13	**17**	**17**
uts-r(16)	**16**	4	1	8	13	12	13	**16**	**16**
total(1179)	784	634	624	702	627	481	473	761	**989**

Table 2: Total number of solved instances across 9 conformant planners. In each row, the number in boldface indicates the best planner. "ER" denotes the planner cannot parse the PDDL file within timeout or encounters an error.

LAMA cannot enumerate every solution with length $2^{2^{|\mathcal{X}|}}$ for the classical planning problem $\mathcal{P}(s)$ since it lacks strong completeness. In addition, our completion procedure cannot extend any solution for $\mathcal{P}(s)$ supplied by LAMA to a correct solution for all initial states. The built-in PDDL file parser of GC used by iGC fails to interpret the PDDL file of the 3 domains adder-ipc5, adder-ipc6 and cleaner as presented in Table 5, and misinterprets some non-mutex propositions into a multi-valued proposition in domain new-uts-cycle.

iGC vs GC: It can be observed from Table 2, iGC successfully solves some instances of the 8 domains: btc, grid-empty and grid-wall, or-1-dis-dis, or-coins, or-dispose, or-new-push and or-push-to of which no instance can be solved by GC. Table 3 highlights some instances with an extremely large number of initial states (at least 65.5K), where GC is unable to find solutions. This is because the initial formula contains no oneof formulas and `oneof`-combination technique is invalid in these instances. In contrast, iGC excels in these instances incredibly quickly since it only deals with a minuscule subset of initial states to derive a solution. For example, in the instance coins-21 with 10^{16} initial states, iGC requires

Instances	$[\![\mathcal{I}]\!]$	GC	iGC
1-dis-dis-4-6	16.8M	TO	0.66/139/15
coins-21	10Q	OM	23.82/1009/84
look-and-grab-10-1-3	1M	TO	22.11/292/11
new-dispose-4-5	1.05M	TO	0.82/252/19
or-1-dis-dis-2-4	65.5K	OM	0.07/12/3
or-coins-10	1M	OM	0.14/58/16
or-dis-3-2	262.1K	OM	0.11/64/17
or-new-push-4-1	65.5K	OM	0.12/34/16
ring-13	20.7M	TO	0.14/59/4
to-trash-12-3	3.0M	TO	210.16/1906/144

Table 3: Some results for instances with a huge number of initial states. Column $[\![\mathcal{I}]\!]$ refers to the number of initial states where the abbreviations K, M and Q represent thousand, million and quadrillion, respectively. "TO" in Column "GC" denotes GC runs timeout for the instance while "OM" denotes GC runs out of memory limit. Column "GC" indicates the solving time (in seconds), the length of solutions and the number of states generated by iGC, respectively.

Domains	Length		iGC/GC
	GC	iGC	
1-dis-dis	496.98	**250.34**	0.50
blocks	336.8	**203.2**	0.60
bomb	80.71	**80.71**	1.00
bt	10.83	**10.83**	1.00
cleaner	**22.5**	23.5	1.04
coins	**129.73**	213.33	1.64
comm	**144.36**	145.84	1.01
cor-sqr	310.63	**308.11**	0.99
cor-cube	297.14	**295.23**	0.99
cube-center	377.59	**234.73**	0.62
dispose	3623.29	**2304.41**	0.64
forest	613.25	**187.25**	0.31
grid	47.67	**35.87**	0.75
hall-a	51.94	**47.06**	0.91
hall-r	90.64	**46.36**	0.51
logistics	100.75	**96.25**	0.96
look-and-grab-dis	211.17	**61.61**	0.29
look-and-grab	242.67	**59.67**	0.25
new-dispose	564	**223.5**	0.40
new-push	281.5	**249**	0.88
new-ring	124.14	**24.29**	0.20
push-to	1980.3	**1338.44**	0.68
raos-key	257	**65**	0.25
retrieve	**598**	1306.36	2.18
reward	**468**	972.5	2.08
ring	**24.25**	26.41	1.09
safe	**44.17**	**44.17**	1.00
sortnet	45.2	**39.73**	0.88
sqr-center	253.45	**188**	0.74
to-trash	3461.6	**1277**	0.37
uts-cycle	**126.57**	229	1.81
uts-k	111.95	**110.11**	0.98
uts-l	940.94	**91.53**	0.10
uts-r	**123.31**	123.5	1.00

Table 4: Average length of solutions in commonly solved instances by GC and iGC.

Instance	Total Time	Solving Time	Parsing Time
adder-ipc5-1	3600	-	3600
adder-ipc6-1	3600	-	3600
bomb-200-200	6.38	1.77	4.61
cleaner-2-20	3600	-	3600
cleaner-5-5	98.04	0.01	98.03
forest-9	42.03	34.52	7.51
look-and-grab-dis-8-3-3	18.32	8.85	9.47
new-dispose-7-7	274.20	20.02	254.18
new-dispose-10-10	430.15	273.51	156.64
new-ring-10	119.89	9.90	109.99
new-ring-11	795.60	169.49	626.11

Table 5: Some instances requiring excessive time for parsing PDDL. Column "Total Time" indicates the overall time, including both parsing and solving phases, "Solving Time" refers to the solving phase and "Parsing Time" refers to the parsing phase.

only 84 initial states to construct a correct solution within 24s. As illustrated in Table 4, iGC produces a solution with average length shorter compared to GC in 21 domains thanks to the elimination of redundant segments in solutions. In particular, for domains 1-dis-dis, forest, look-and-grab-dis, look-and-grab, new-ring, raos-key, to-trash and uts-l, iGC compute the solutions with length only half of those generated by GC. iGC generates solutions that are at least 60% longer than those of GC in the 4 domains: coins, retrieve, reward and uts-cycle. This is because the completion procedure in the original GC generates more succinct solutions than the improved planner. In summary, iGC demonstrates clear advantages over GC in both the number of solved instances and the quality of solutions across most domains.

iGC vs CPCES: As shown in Table 6, with the exception of grid-wall, new-push, new-ring, and safe, iGC takes less time on average for verification than iCPCES and gCPCES. For example, in cube-center, iGC and iCPCES solve problems with comparable average iterations and solution lengths. However, iGC completes verification in only 0.31s while iCPCES takes 3.99s. In or-dispose, although iGC uses 118.5 iterations on average, while iCPCES has 41.69 iterations, but the verification time of the former is significantly faster than that of the latter (1.62s versus 41.29s). This is due to our simplification strategy on the trace axioms and elimination strategy on solutions in iGC which effectively reduces the number of propositions and the length of the trace axiom, thereby accelerating the SAT-based verification procedure. However, iGC performs worse than CPCES in the four domains: cleaner, new-push, new-ring and safe. In cleaner and new-ring, iGC takes a majority of time in parsing PDDL file. For example, the cleaner-5-5 instance takes 98.03s to parse but only 0.01s to solve, while new-ring-10 requires 109.99s for parsing and just 9.90s for solving. In addition, some instances requiring excessive time for parsing PDDL in iGC are shown in Table 5. In the two domains: new-push and new-ring, iGC requires much more iterations than iCPCES to construct the solution. For example, in new-push-4-4 and new-ring-7, iGC requires 2871 and 1066 iterations while gCPCES needs only 13 and 2 iterations, respectively. In the safe domain, the number of propositions in the trace axiom generated by CPCES is only half of that in iGC due to different PDDL file parsers used by iGC, iCPCES and gCPCES. Overall, iGC is faster than both iCPCES and gCPCES in the majority of domains.

Domains	Total Time			Verification Time		
	iCPCES	gCPCES	iGC	iCPCES	gCPCES	iGC
1-dis-dis	7.97	0.29	**0.07**	0.36	0.07	**0.01**
blocks	0.65	0.30	**0.08**	0.08	0.07	**0.01**
bomb	NA	303.77	**0.17**	NA	7.52	**0.02**
bt	0.72	0.36	**0.08**	0.09	0.10	**0.03**
btc	0.73	0.45	**0.09**	0.11	0.12	**0.05**
cleaner	**0.29**	0.44	33.72	0.03	0.19	**0.01**
coins	6.20	5.18	**0.72**	3.93	1.02	**0.46**
comm	NA	49.17	**0.19**	NA	14.78	**0.07**
cor-sqr	179.99	NA	**1.39**	14.16	NA	**1.14**
cor-cube	225.73	NA	**0.27**	1.96	NA	**0.17**
cube-center	302.88	144.39	**0.41**	3.99	4.58	**0.31**
dispose	35.79	257.36	**2.14**	33.56	19.52	**1.79**
forest	0.47	0.28	**0.12**	0.02	0.05	**0.01**
grid	0.55	0.36	**0.09**	0.14	0.11	**0.02**
grid-empty	NA	0.37	**0.08**	NA	0.09	**0.04**
grid-wall	NA	50.41	**7.04**	NA	**2.03**	6.22
hall-a	0.32	0.93	**0.07**	0.02	0.09	**0.01**
hall-r	0.48	NA	**0.11**	0.06	NA	**0.03**
logistics	0.83	1.91	**0.22**	0.15	0.30	**0.06**
look-and-grab-dis	161.75	NA	**0.40**	0.50	NA	**0.09**
look-and-grab	0.27	0.22	**0.06**	0.01	0.05	**0.00**
new-dispose	554.42	NA	**114.88**	454.95	NA	**54.43**
new-push	8.41	**4.94**	15.42	1.31	**0.71**	11.34
new-ring	NA	**0.23**	135.91	NA	**0.07**	32.92
or-dis-dis	NA	0.32	**0.06**	NA	0.09	**0.01**
or-coins	7.62	7.51	**1.06**	4.81	1.35	**0.44**
or-dispose	43.28	116.20	**6.07**	41.29	11.17	**1.62**
or-new-push	1.53	2.21	**0.17**	0.53	0.48	**0.06**
or-push-to	7.14	34.57	**0.26**	2.60	0.78	**0.10**
push-to	188.27	44.69	**0.13**	1.07	1.21	**0.05**
raos-key	6.72	3.46	**0.14**	1.42	1.32	**0.07**
retrieve	44.53	457.85	**1.81**	41.06	70.27	**1.53**
reward	175.58	626.64	**3.24**	62.93	98.89	**2.75**
ring	0.34	0.24	**0.05**	0.02	0.05	**0.01**
safe	7.40	**2.05**	4.48	5.13	**1.23**	4.21
sqr-center	366.97	3.59	**1.85**	14.31	2.16	**1.54**
to-trash	133.54	NA	**1.54**	28.84	NA	**1.30**
uts-k	233.38	6.79	**0.59**	3.35	1.60	**0.36**
uts-l	18.55	121.14	**0.80**	8.36	4.19	**0.67**
uts-r	358.48	23.08	**1.54**	7.53	2.72	**1.22**

Table 6: Average solving and verification time of iGC, iCPCES and gCPCES in commonly solved domains. If iCPCES cannot solve any instance for a domain where both gCPCES and iGC solves at least one common instance, then the results for that domain are still included, and "NA" indicates the failure of iCPCES. The same applies to gCPCES.

5.3 Ablation Study

Table 7 shows the results about the 42 domains solvable by iGC with and without optimizations. We use iGC for the iGC approach without simplification on trace axioms and elimination of redundant fragements, iGC+S for iGC with only simplification, iGC+E for iGC with only elimination and iGC+SE for iGC with both optimizations. For iGC+S, 14 domains have an increase of 50 solved instances relative to iGC. However, 7 domains experience a reduction of 31 solved instances, with notable decreases observed in the 1-dis-dis and new-push domains, where the number of solved instances drops by 18 and 6, respectively. Totally, iGC+S increases the total number of solved instances by 19. In comparison, iGC+E improves performance in 13 domains, yielding an additional 64 solved instances, while three domains show a reduction of 13 solved instances. In summary, iGC+E solves 51 more instances than iGC without optimizations. When both optimizations are applied together, 14 domains collectively achieve an increase of 73 solved instances, whereas

Domains	Coverage				Total Time				Length				Verificate Time			
	iGC	iGC+S	iGC+E	iGC+SE	iGC	iGC+S	iGC+E	iGC+SE	iGC	iGC+S	iGC+E	iGC+SE	iGC	iGC+S	iGC+E	iGC+SE
1-dis-dis(90)	**70**	52	61	65	47.03	15.27	17.93	**13.55**	326.79	591.36	**181.55**	250.89	43.20	12.01	10.47	**6.77**
blocks(5)	**5**	**5**	**5**	**5**	**6.48**	7.75	54.67	62.05	328.00	318.80	212.80	**203.20**	**5.74**	7.26	34.26	40.55
bomb(21)	**21**	**21**	**21**	**21**	0.64	0.55	0.68	**0.52**	**80.71**	**80.71**	**80.71**	**80.71**	0.15	**0.04**	0.15	**0.04**
bt(19)	**19**	**19**	**19**	**19**	0.09	**0.08**	**0.08**	**0.08**	**11.00**	**11.00**	**11.00**	**11.00**	0.04	0.04	0.04	**0.03**
btc(19)	**19**	**19**	**19**	**19**	0.10	**0.09**	0.10	**0.09**	**21.00**	**21.00**	**21.00**	**21.00**	0.06	**0.05**	0.06	**0.05**
cleaner(10)	**3**	**3**	**3**	**3**	35.01	35.34	33.37	**33.72**	**26.00**	26.33	27.67	27.00	**0.01**	**0.01**	**0.01**	**0.01**
coins(25)	**25**	**25**	**25**	**25**	17.33	58.43	**12.95**	44.85	725.60	1414.76	**674.04**	926.24	15.47	57.07	**11.38**	33.90
comm(25)	**25**	**25**	**25**	**25**	0.78	0.23	0.52	**0.19**	150.04	150.04	**145.84**	**145.84**	0.61	0.11	0.36	**0.07**
cor-sqr(19)	**19**	**19**	**19**	**19**	1.44	**1.39**	1.49	**1.39**	308.11	308.11	**308.11**	**308.11**	1.18	**1.12**	1.20	1.14
cor-cube(22)	**22**	**22**	**22**	**22**	2.51	**2.14**	2.54	2.19	301.55	301.55	295.36	**295.23**	2.20	**1.87**	2.22	1.93
cube-center(37)	35	**37**	**37**	**37**	102.62	3.91	33.96	**3.43**	892.43	244.77	**219.34**	**219.34**	96.83	3.36	30.02	**2.88**
dispose(98)	81	**97**	90	**97**	258.31	20.76	94.22	**19.93**	1615.12	1615.12	**877.84**	1291.47	243.50	20.04	85.59	**17.64**
forest(8)	**8**	**8**	**8**	**8**	**15.36**	17.01	16.44	17.01	189.63	188.88	188.75	**187.25**	0.09	**0.03**	0.10	0.04
grid(15)	13	**15**	**15**	**15**	5.02	**0.08**	**0.08**	**0.08**	292.62	54.54	30.23	**29.46**	4.20	**0.02**	**0.02**	**0.02**
grid-empty(18)	**4**	**4**	**4**	**4**	0.11	**0.08**	0.09	**0.08**	44.50	42.50	22.50	**18.50**	0.07	**0.04**	**0.04**	**0.04**
grid-wall(18)	8	17	**18**	**18**	0.46	**0.42**	1.35	0.60	111.75	103.00	**48.50**	55.25	0.41	**0.37**	1.14	0.50
hall-a(21)	**19**	18	**19**	18	0.20	0.21	0.19	**0.18**	**47.06**	**47.06**	**47.06**	**47.06**	0.06	0.06	0.07	**0.05**
hall-r(11)	**11**	**11**	**11**	**11**	**1.14**	1.32	1.28	1.34	**46.36**	**46.36**	**46.36**	**46.36**	**0.05**	0.06	0.08	0.10
logistics(17)	15	**16**	**16**	**16**	14.81	0.71	0.50	**0.35**	227.07	230.60	72.67	**72.40**	11.41	0.37	0.28	**0.13**
look-and-grab-dis(18)	**18**	**18**	**18**	**18**	**4.10**	8.80	7.05	5.43	93.50	108.22	**59.11**	61.61	1.03	0.86	0.76	**0.69**
look-and-grab(80)	**80**	**80**	**80**	**80**	**3.47**	14.88	12.66	15.86	96.36	108.83	**60.96**	62.58	2.19	1.35	2.16	**1.32**
new-dispose(6)	**6**	5	**6**	**6**	862.98	297.54	336.89	**146.07**	2699.80	2676.60	**1193.40**	1440.20	791.81	149.43	280.21	**98.56**
new-push(12)	**12**	6	9	8	28.71	269.74	**10.36**	11.28	1758.83	1986.33	**150.50**	185.50	25.25	264.50	8.65	**8.19**
new-ring(11)	**9**	**9**	**9**	**9**	50.54	**18.00**	39.37	135.91	283.33	91.00	38.67	**35.44**	33.15	**1.85**	11.57	32.92
or-1-dis-dis(16)	12	9	**15**	11	3.78	**0.92**	3.48	1.49	88.00	87.00	**69.75**	71.00	3.36	0.74	2.42	**0.64**
or-coins(10)	9	**10**	8	9	126.28	**21.55**	84.98	102.52	1348.71	940.14	1334.00	**781.14**	114.39	19.97	51.77	**14.84**
or-dispose(23)	16	**19**	16	16	342.34	**35.82**	406.54	238.50	1308.13	1303.13	**894.63**	904.13	316.31	34.22	193.50	**24.65**
or-new-push(20)	9	8	11	**12**	3.22	**1.10**	247.76	1.64	106.14	108.29	**65.67**	70.43	2.79	0.85	1.46	**0.58**
or-push-to(16)	8	**9**	8	8	47.81	**10.33**	42.07	21.40	404.00	424.75	**292.63**	304.13	42.41	8.50	23.37	**5.75**
push-to(108)	101	**108**	106	**108**	201.29	29.66	122.00	**24.98**	1150.78	1755.31	**830.70**	1097.18	189.49	27.01	112.29	**21.58**
raos-key(8)	**2**	**2**	**2**	**2**	0.21	**0.14**	0.24	**0.14**	83.50	88.00	**62.50**	65.00	0.16	0.09	0.16	**0.07**
retrieve(15)	13	**14**	13	**14**	498.02	**172.45**	442.84	190.13	690.92	988.54	**675.62**	985.23	463.09	**164.34**	402.20	170.89
reward(15)	11	**12**	11	**12**	437.84	**327.21**	418.81	329.60	451.45	759.91	**443.55**	776.64	412.83	313.76	389.49	**304.47**
ring(100)	**100**	**100**	**100**	**100**	5.67	6.27	5.58	**5.16**	277.61	298.73	**265.86**	270.67	5.25	5.95	5.14	**4.85**
safe(6)	**6**	**6**	**6**	**6**	4.69	**4.45**	4.72	4.48	**44.17**	**44.17**	**44.17**	**44.17**	4.44	4.23	4.43	**4.21**
sortnet(15)	**15**	**15**	**15**	**15**	0.65	**0.35**	21.80	25.09	67.87	58.20	**39.73**	**39.73**	0.55	**0.23**	6.22	3.72
sqr-center(30)	**30**	**30**	**30**	**30**	149.72	**3.53**	32.41	3.66	945.43	223.37	**183.23**	182.87	141.78	**3.01**	28.43	3.10
to-trash(15)	8	11	**13**	12	68.61	47.41	41.43	**30.33**	502.50	1188.75	**343.25**	602.50	63.00	44.84	35.79	**27.07**
uts-cycle(28)	12	13	13	**14**	130.22	244.75	**28.92**	36.11	303.00	569.25	**144.75**	158.42	129.90	244.23	28.75	**35.90**
uts-k(19)	11	10	**19**	**19**	2.47	11.84	0.19	**0.15**	65.78	66.56	28.11	**28.00**	2.21	3.11	0.11	**0.07**
uts-l(17)	8	10	**17**	**17**	5.08	29.43	0.12	**0.10**	157.63	151.13	**25.00**	**25.00**	4.76	28.88	0.07	**0.05**
uts-r(16)	9	9	**16**	**16**	4.13	7.30	0.24	**0.19**	99.89	97.22	37.56	**37.33**	3.85	3.07	0.16	**0.12**
total(1102)	927	946	978	**989**	-	-	-	-	-	-	-	-	-	-	-	-

Table 7: Comparisons of iGC with/without simplification and/or elimination optimizations. The reported results are the average value of all instances that can be solved by all variants of iGC in each domain.

three domains demonstrate a reduction of 10 instances. The total number of solved instances rises by 62, showing the complementary nature of these two optimizations.

Regarding solving time, iGC+S exhibits superior performance across 28 domains by simplifying trace axioms, which significantly reduce verification time. Similarly, iGC+E is faster than iGC in 27 domains. When both optimizations are employed, the approach iGC+SE is faster in 33 domains compared to iGC. However, performance degradation is observed in certain domains, primarily due to the counterexamples returned by the SAT solver. Different trace axioms generates different initial states, and hence affects the overall performance of iGC planner. For instance, in the 1-dis-dis domain, instances solvable by iGC fail with iGC+S due to memory overflow during the subproblem completion process.

In terms of solution length, iGC+E effectively eliminates redundant actions in 34 domains. Notably, 76.5% of the domains achieve length reductions exceeding 20%. The new-push domain demonstrates the most substantial improvement, with a 91.4% reduction in solution length. These results undercover significant enhancements in solution quality. Although simplifying optimization sometimes increases solution lengths, the combination with elimination optimization mitigates this issue.

6 Conclusions

In this paper, we have observed two major drawbacks of the GC approach: the computational overhead due to state exploration and the insertion of many redundant actions. To solve these two drawbacks, we have proposed SAT-based verification procedure and counterexample-guided completion procedure, and have integrated them into the GC approach, resulting in an improved GC planner, namely iGC. We have demonstrated the superior performance and scalability of iGC by evaluating it on standard benchmarks for conformant planning. In particular, iGC is the top planner in terms of coverage, solving 989 out of 1179 instances. Compared with the original GC planner, iGC exhibit the quality advantage of solutions and the ability to solve instances with a huge number of initial states.

Acknowledgements

We are grateful to Mingwei Zhang for his constructive comments on the paper. We sincerely thank Zexin Cai, Yuchun Zhong and Sijie Yin for their contributions to collecting and modifying PDDL files of each instance used in the experiments. This paper was supported by National Natural Science Foundation of China (Nos. 62206110, 62276114 and 62377028), Guangdong Basic and Applied Basic Research Foundation (Nos. 2023B1515120064 and 2024A1515011762), Guangzhou Science and Technology Planning Project (Nos. 202206030007, 2025A03J3565, Nansha District: 2023ZD001 and Development District: 2023GH01) and Major Key Project of PCL (Nos. PCL2024A04 and PCL2024AS204).

References

[Albore *et al.*, 2011] Alexandre Albore, Miquel Ramirez, and Hector Geffner. Effective Heuristics and Belief Tracking for Planning with Incomplete Information. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS-2011)*, pages 2–9, 2011.

[Angluin, 1987] Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[Bonet and Geffner, 2000] Blai Bonet and Hector Geffner. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS-2000)*, pages 52–61, 2000.

[Bonet and Givan, 2006] Blai Bonet and Bob Givan. Results of the Conformant Track of the 5th Planning Competition, 2006.

[Bryce and Buffet, 2008] Daniel Bryce and Olivier Buffet. The Uncertainty Part of the 6th International Planning Competition, 2008.

[Bryce *et al.*, 2006] Daniel Bryce, Subbarao Kambhampati, and David E. Smith. Planning Graph Heuristics for Belief Space Search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[Cimatti and Roveri, 2000] Alessandro Cimatti and Marco Roveri. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[Cimatti and Roveri, 2004] Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159(1-2):127–206, 2004.

[Clarke *et al.*, 2003] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM*, 50(5):752–794, 2003.

[de Moura and Bjørner, 2008] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Proceedings of the Fourteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-2008)*, pages 337–340, 2008.

[Eiter *et al.*, 2003] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A Logic Programming Approach to Knowledge-state Planning, II: The DLVK System. *Artificial Intelligence*, 144(1-2):157–211, 2003.

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

[Haslum and Jonsson, 1999] Patrik Haslum and Peter Jonsson. Some Results on the Complexity of Planning with Incomplete Information. In *Proceedings of the Fifth European Conference on Planning (ECP-1999)*, volume 1809 of *Lecture Notes in Computer Science*, pages 308–318. Springer, 1999.

[Hoffmann and Brafman, 2006] Jörg Hoffmann and Ronen I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170:507–541, 2006.

[Nguyen *et al.*, 2011] Khoi Nguyen, Vien Tran, Tran Cao Son, and Enrico Pontelli. On Improving Conformant Planners by Analyzing Domain-structures. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-2011)*, pages 998–1003, 2011.

[Nguyen *et al.*, 2012] Khoi Nguyen, Vien Tran, Tran Cao Son, and Enrico Pontelli. On Computing Conformant Plans Using Classical Planners: a Generate-And-Complete Approach. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-2012)*, pages 190–198, 2012.

[Palacios and Geffner, 2009] Hector Palacios and Hector Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

[Rintanen, 2002] Jussi Rintanen. Backward Plan Construction for Planning with Partial Observability. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-2002)*, pages 173–183, 2002.

[Smith and Weld, 1998] David E. Smith and Daniel S. Weld. Conformant Graphplan. In *The Fifteenth National Conference on Artificial Intelligence (AAAI-1998)*, pages 889–896, 1998.

[To *et al.*, 2015] Son Thanh To, Tran Cao Son, and Enrico Pontelli. A Generic Approach to Planning in the Presence of Incomplete Information: Theory and Implementation. *Artificial Intelligence*, 227:1–51, 2015.

[Tran *et al.*, 2013] Vien Tran, Khoi Nguyen, Tran Cao Son, and Enrico Pontelli. A Conformant Planner Based on Approximation: CpA(H). *ACM Transactions on Intelligent Systems and Technology*, 4(2):36:1–36:38, 2013.