Counter-Example Based Planning – Dissertation Abstract

Xiaodi Zhang[1]

Supervisors: Alban Grastien[1]**, Hanna Kurniawati**[1]**, Charles Gretton**[1]

[1] School of Computing, Australian National University
Acton, ACT, Australia, 2601
firstname.lastname@anu.edu.au

Abstract

`CPCES` is one of conformant planning problem solvers. It continuously searches candidate plans and counter-examples until finding a valid plan or no solution. The goal of my Ph.D. project includes improving its efficiency, making it compatible with more classical planners, and using it to solve contingent planning problems.

Background

Conformant Planning Problem

Conformant planning is the problem of searching for a plan in an environment that is partially known: either the initial state or the actions' effects are non-deterministic. Conformant planning problem is EXPSPACE-complete (Haslum and Jonsson 1999). In comparison, classical planning problem with no uncertainty is only PSPACE-complete. Conformant planning can be used in different areas. For example, Mars exploration robot can collect soil samples in an unknown environment; logistics company distributes trunks and airplanes in different cities; etc.

Given a set of facts V, a state s is a set of facts s. A conformant planning model is defined as a tuple $P = \langle V, A, \Phi_I, \Phi_G \rangle$, where

- V is a finite set of facts
- Φ_I is the initial states, $s \models \Phi_I$
- Φ_G is the goal states, $s \models \Phi_G$
- A is a set of actions. Each action $a \in A$ is defined as a pair $\langle pre, coneff \rangle$, where pre is the precondition, and $coneff$ is a set of conditional effects. Each conditional effect is a pair $\langle c, eff \rangle$, where c is the condition, and eff is the effects. Effect includes positive effects eff^+ and negative effect eff^-. If a state s satisfies the precondition, after executing action a, the next state s' is $s' = s \cup (\bigcup_{\langle c, eff \rangle \in coneff, s \models c} eff^+) \backslash (\bigcup_{\langle c, eff \rangle \in coneff, s \models c} eff^-)$.

The solution π to P is a sequence of actions $\pi = a_1 ... a_n$. A valid plan must reach the goal states for all initial states.

Here is an example of conformant planning problem in Figure 1. At the beginning, a robot is located in a 5×5 grid, but its exact location is unknown. The robot has access to four actions: GO_E, GO_W, GO_S, and GO_N (go East, go West, go South, and go North, respectively). The grid is surrounded by walls to prevent the robot from moving out. In other words, when robot hits the wall, it keeps its position. The robot should finally move to (3,3), denoted as "G". The unknown initial state is the trickiest part of this problem. However, if the robot does GO_E $\times$ 4, GO_S $\times$ 4, no matter where it is at beginning, it must be finally at (5,1), denoted as "T". So, one of the solutions is GO_E $\times$ 4, GO_S $\times$ 4, GO_N $\times$ 2, GO_W $\times$ 2.

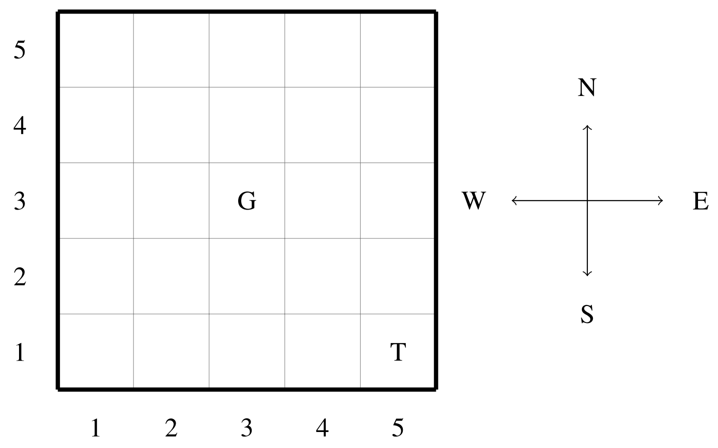

Figure 1: A robot is required to move to "G" but the initial state is unknown. Doing GO_E $\times$ 4, GO_S $\times$ 4 will move the robot to "T" no matter where it is at beginning, since the robot keeps standing when it hits the wall. One of solutions is GO_E $\times$ 4, GO_S $\times$ 4, GO_N $\times$ 2, GO_W $\times$ 2.

Literature Review

It is impractical to enumerate all the possible states during the search, so how to represent belief states decides the efficiency of a planner, and the way of representation is the most difficult parts when designing a planner. `T1` is translation-based approach (Albore, Ramirez, and Geffner 2011). It addresses the difficulty by compiling planning problem to another one which adopts a more compact representation of belief states, so it is easier to be solved. Currently, `T1` is one of the best conformant planners. `Conformant-FF` selects a specific language to represent the problem and uses heuristic functions to estimate the distance to the goal (Hoffmann and Brafman 2006). But `conformant-FF` has two short comings. First, when there are effects with more than one unknown condition, `conformant-FF` cannot get enough heuristic values. Second, it is expensive in checking repeated states in the problem of lacking different known propositions in the belief states. `MBP` is a model-based approach. It uses Symbolic Model Checking techniques and Binary Decision Diagrams to handle a large size of belief states (Bertoli et al. 2001). The shortcoming of `MBP` is that it requires high memory. `POND` chooses to use Labeled Uncertainty Graph (LUG) to represent GraphPlan-like information for different initial states, and use lazily enforced hill-climbing strategy to search a plan (Bryce 2006). The `CNF` planner uses a specific form of CNF, CNF-state, to represent belief states and solve the problem (To, Son, and Pontelli 2010), but for some benchmarks, it spends long time in searching a solution.

As we can see, most planners represents belief states by using other form of languages, diagrams or models. Scala and Grastien created a new method, `CPCES`, which simplifies the problem not through a specific representation, but through searching counter-examples (Grastien and Scala 2020).

CPCES

`CPCES` at first was designed to solve deterministic conformant planning problem (unknown initial states and deterministic actions' effects). The algorithm of `CPCES` can be explained as follow: `CPCES` addresses the problem by replacing belief states with a small set of initial states, which is called sample. In each iteration, `CPCES` searches a candidate plan that is valid for the sample. If no candidate plan exists, there is no solution to the problem. Then, `CPCES` searches a counter-example, an initial state that breaks the candidate plan. If no such counter-example exists, the candidate plan is a valid plan. Otherwise, the counter-example is added into the sample (Algorithm 1).

Algorithm 1: The conformant planner `CPCES`.

```
1: input: conformant planning problem $P$
2: output: a conformant plan, or no plan
3: $B := \emptyset$
4: loop
5:     $\pi :=$ produce-candidate-plan$(P, B)$
6:     if there is no such $\pi$ then
7:         return no plan
8:     end if
9:     $q :=$ generate-counter-example$(P, \pi)$
10:    if there is no such $q$ then
11:        return $\pi$
12:    end if
13:    $B := B \cup \{q\}$
14: end loop
```

`CPCES` consists of two main parts. The first part (Line 9) uses SMT formula to search a counter-example to a candidate plan. The basic idea of search counter-example is to do regression, finding an initial state that breaks the precondition of an action or dissatisfies the goal. In the second part (Line 5), `CPCES` translates the conformant planning problem to the classical planning problem by creating a multi-interpretation domain and a multi-interpretation instance (both are PDDL files), then uses `Fast Forward (FF)` (Hoffmann and Nebel 2001) to search a candidate plan. The translation procedure is named the reduction of conformant planning: executing n classical plannings in parallel where all actions are synchronized. Because of the reduction, the candidate plan must be valid for all samples.

`CPCES` has been proved sound and complete. It only uses a small set of initial states to search candidate plans, significantly decreasing the cost. Compared with `T1`, `CPCES` performs better on complex domains while `T1` is good at 1-width domains (the conformant width is the maximum number of facts in the context whose initial value is unknown).

Scala and Grastien further developed `CPCES` to address non-deterministic conformant planning problems (Scala and Grastien 2021). Non-deterministic conformant planning is a more complex issue in which both initial states and actions' effects are uncertain. Using Figure 1 as an example for non-deterministic planning. Now the robot is broken. When it wants to move East, it may move North-East instead. To solve this kind of problem, `CPCES` translates a non-deterministic conformant planning to a classical planning by synchronizing this problem with a non-deterministic finite automaton (NFA) that prevents the invalid solutions.

Work I Have Done

Improve CPCES by Searching Superior Counter-Examples

I noticed that `CPCES` searches counter-examples randomly. Is it possible to improve `CPCES` by searching a specific kind of counter-examples? Look at Figure 1. Now there are two robots in the grid, and both of them are required to move to "G". We suppose robots do not collide when they are at the same position. In Table 1, I listed several conditions of counter-examples. The first counter-example (CE1) for two robots are (1,1), and the second counter-example (CE2) for Robot2 is (5,5) but Robot1 remains (1,1). The candidate plan generated in the 2nd round may be weak, because `CPCES` considers Robot1 can only be at (1,1). Now we update CE2 to CE2' where Robot1 is at (4,4). The candidate plan becomes stronger, because both (1,1) and (4,4) are covered for Robot1. I was inspired from this example, believing that the more information the counter-example covers, the more efficient `CPCES` searches a valid plan. So I developed `CPCES` by searching a superior counter-example over another one in each round. A superior counter-example is computed by $tags$.

	Robot1	Robot2
CE1	(1,1)	(1,1)
CE2	(1,1)	(5,5)
CE2'	(4,4)	(5,5)

Table 1: If CE2 for Robot1 remains the same as CE1, it is not a good counter-example. Updating CE2 to CE2' in which Robot1 is located at (4,4) will be better.

A *subgoal* φ is a conjunct of an action precondition or the goal. An action a can be applied at state s only when $\varphi(s)$ is true. A variable v *depends on* another variable v' if there exists an action a that mentions variable v' in its conditional effect. The *context* $ctx(\varphi)$ of subgoal φ is a set of variables mentioned by φ, the variables they depend on, and by transitivity, all the variables they depend on. Given a context c, a *tag* for an initial state s is denoted as $tag_c(s)$, which is the intersection of c and s: $tag_c(s) = c \cap s$. Given a belief state B and a context c, the set of tags of B for c is $T_c(B) = \{tag_c(s) | s \in B\}$. We use $T(B) = \bigcup_{c \in C} T_c(B)$ to represent all the tags in the problem, where C is a set of all the contexts.

Algorithm 2: `compute-superior-counter-example`

```
input: conformant planning problem $P$
$q$ := generate-counter-example($\pi$)
if there is no such $q$ then
    return $\pi$ is valid
end if
loop
    $q'$ := improve-counter-example($B, q$)
    if there is no such $q'$ then
        return $q$
    end if
    $q := q'$
end loop
```

A counter-example s' is better than another counter-example s if $T(B \cup s) \subseteq T(B \cup s')$. This is because the more tags there are, the more information we get. The intersection of the set of plans generated from each tag is the same as the plans generated from belief state: $\Pi(B) = \bigcap_{t \in T(B)} \Pi(t)$. The experiment results illustrate that finding superior counter-examples makes CPCES more efficient on multi-context domains, while on 1-context domains the efficiency is not impacted. This work has been published at AAAI-2020 (Zhang, Grastien, and Scala 2020).

Use Fast Downward Planner in CPCES

CPCES uses FF (Helmert 2006) to search candidate plans. I considered adding Fast Downward (FD) into CPCES as an option of planner, since FD provides more options on search engines and heuristic functions.

FD contains two parts (Helmert 2006). The first part translates PDDL files to a SAS+ file. It computes mutex groups, translates the problem to a multi-valued planning task, and finally writes a SAS+ file to save the task. The second part is using SAS+ file to search a valid plan. Translation helps FD simplify the question so the problem can be solved quickly. However, this is a trade-off, since translation itself is a complex procedure and it is expensive.

Using FD in CPCES directly is impractical. CPCES calls FF in each iteration, and with the number of iterations increasing, the interpretation instance file becomes larger and larger. It is difficult for FD to handle with translating a large file, not to mention FD will be used in all iterations. I did some experiments and the results display that for all the domains, the searching time of FD is too long, sometimes even out of memory.

To address this issue, I designed a viable approach to use FD in CPCES. In CPCES's interpretation instance file, each state variable v is replaced by a new variable v_i where i is the interpretation number (iteration number). It is easy to understand that if two variables v_i and v'_i are mutually exclusive, v_k and v'_k ($k \neq i$) must be mutually exclusive. This is because variables represented by k and variables represented by i are two parallel classical problems but have the same properties. So separately translating interpretation file and merging them is applicable.

Having FD translate the whole interpretation file is difficult, but translating an interpretation file with only single interpretation number should be much easier. Based on this idea, I designed a new algorithm to replace FF with FD by letting FD translate the interpretation file with only one interpretation number in each iteration, and merging the result into previous SAS+ file (Figure 2).

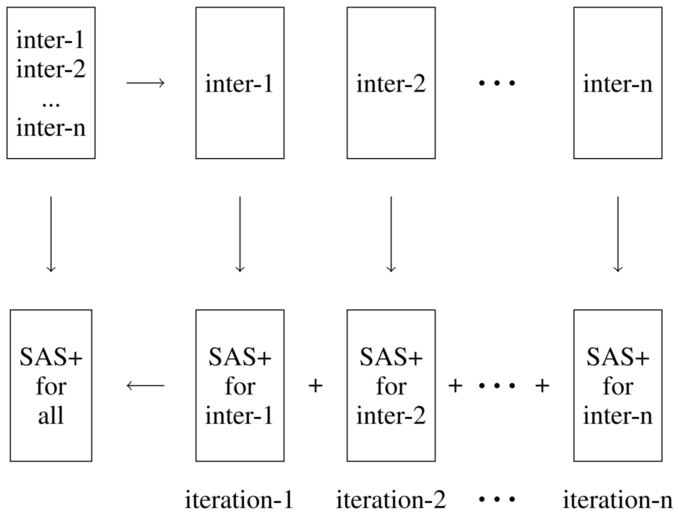

Figure 2: In each iteration, CPCES adds a new interpretation variable into interpretation instance file, so the interpretation instance file becomes larger and larger. Translating a large PDDL file to a SAS+ file by FD is impractical. I solved this problem by asking FD to translate the interpretation instance file with a single interpretation number in each iteration, then merge the result with previous SAS+ file.

This approach has been tested on several benchmarks. The search engine in FD is eager best-first search, with best-first open list and FF heuristic. The results shows that the search time is extremely shorter than using FD directly. However, this approach is currently worse than FF in terms of searching time. I am not sure whether this is because the search engine is not the best. So, I will test more search engines in the future.

Future Works

Add More Planners into CPCES

`CPCES` currently chooses `FF` to search a plan. This is not friendly, because some people may be interested in other planners or algorithms, such as `FD`, granplan, partial order planning. It is necessary to add them into `CPCES`.

Contingent Planning Problem

In contingent planning, the initial state is unknown, but the agent can observe the nearby environment during the actions. For example, in Figure 1, when the robot goes to "T", the wall at the East and the South will be detected. Then, the robot will GO_N or GO_W rather than GO_E or GO_W to avoid hitting the wall.

Given a set of facts V, a state s is a set of facts $s(v)$ where $v \in V$. A contingent planning model is defined as a tuple $P = \langle V, A, O, \Phi_I, \Phi_G \rangle$, where

- V, Φ_I, Φ_G, and A are defined as before.
- O is a set of observations (sensing actions). Each observation $o \in O$ can be defined as a pair $\langle c, v \rangle$, where c is the condition, and v is a positive variable in V. The pair indicates that the truth value of v is observable when c holds.

The solution π to P is a tree of actions, in which the fork is a sensing action, and the child node is the next action decided by what has been observed.

Compared with contingent planning problem, conformant planning problem lacks sensing actions. Because of it, conformant planning is actually a special contingent planning where there is no observation, so no branch in the plan. While `CPCES` works on conformant planning, why not contingent planning?

Increase the Efficiency of CPCES

The number of initial states in `CPCES` decides the complexity of the problem. In fact, some initial states are more important than others. Back to Figure 1, for instance, if we only consider two initial states: (1,1), (5,5), the result is the same as considering all the initial states. These two initial states are called "important initial states". Even though we ignoring all the other 23 initial states, these 2 initial states are strong enough to help us find a valid plan. Finding important initial states before searching a plan may dramatically increase the efficiency of the planner. But how to find these important initial states is a difficult problem. What is more, is the procedure of finding important initial states expensive? Is it worthwhile to find important initial states?

Probabilistic Planning Problem

Uncertainty about action's effect is a key feature of probabilistic planning problem: a probability will be assigned to each effect. As a consequence, the goal is to find a plan that is valid with a specified problem. How to develop `CPCES` to solve probabilistic planning problem? Is `CPCES` more efficient comparing other probabilistic planners? Can I solve probabilistic planning problem by combining `CPCES` with other planners?

Build CPCES Website

`CPCES` is a successful conformant planner, and it has a huge potential to be further developed. Propagating `CPCES` to the world is my duty, so build a website is necessary. We encourage everyone to participate in developing `CPCES` so that `CPCES` can become one of the most useful planners in the world.

References

Albore, A.; Ramirez, M.; and Geffner, H. 2011. Effective heuristics and belief tracking for planning with incomplete information. In *Twenty-First International Conference on Automated Planning and Scheduling*.

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. MBP: a model based planner. In *Proc. of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*.

Bryce, D. 2006. POND: The partially-observable and nondeterministic planner. *Sixteenth International Conference on Automated Planning and Scheduling*, 58.

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

Haslum, P.; and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *European Conference on Planning*, 308–318. Springer.

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

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

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

Scala, E.; and Grastien, A. 2021. Non-Deterministic Conformant Planning Using a Counterexample-Guided Incremental Compilation to Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 299–307.

To, S. T.; Son, T. C.; and Pontelli, E. 2010. A New Approach to Conformant Planning Using CNF*. In *Twentieth International Conference on Automated Planning and Scheduling*.

Zhang, X.; Grastien, A.; and Scala, E. 2020. Computing superior counter-examples for conformant planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 10017–10024.