Conformant Planning via Classical Planners

Khoi Nguyen
Computer Science Department
New Mexico State University
Las Cruces, NM 88001
Email: knguyen@cs.nmsu.edu

Advisor: Dr. Tran Cao Son

Introduction

Conformant planning is the problem of computing a sequence of actions that achieves a goal in presence of incomplete information about the initial state (Smith and Weld 1998). By definition, conformant planning searches for the plan in the belief state space. Due to the incomplete information, the belief state usually has large size which leads to difficulty in searching for the solution. Thus there are two trends in developing conformant planners. The first group of planners employ different compact representations for belief state and develop different heuristics to guide the search in belief state space. That can be observed in POND (Bryce *et al.* 2006), CPA (Tran *et al.* 2009), DNF (To *et al.* 2009) and CNF (To *et al.* 2010). The second group translate the conformant planning problem to another equivalent problem as can be seen in KACMBP (Cimatti *et al.* 2004) and `t0` (Palacios and Geffner 2006). In this group, `t0` has exceptional performance due to its approach of translating the conformant planning problem to classical planning problem.

The idea of using a classical planning system to solve a non-classical planning problem has been applied to other types of planning problems such as probabilistic planning. FF-Replan (Yoon *et al.* 2007), the winner of the 2004 IPC, solves a probabilistic planning problem by (*i*) translating the problem into a classical planning problem, (*ii*) computing a solution using a classical planner (FF), and (*iii*) replanning whenever necessary.

It is interesting to contrast the approaches adopted in `t0` and FF-Replan. While the translation employed by `t0` could produce a new problem whose size is exponential in the size of the original one (if completeness is required), and thus making the problem more difficult, the *determinizing* process of FF-Replan simplifies the original problem by removing all information related to non-determinicity. This raises the interesting question of whether an alternative approach to `t0`, perhaps in a similar spirit to that of FF-Replan, could produce similar results in conformant planning. It is clear that the algorithm of FF-Replan cannot be applied to conformant planning, since conformant planning does not interleave planning and execution.

My thesis aims at investigating the aforementioned question. We want to develop a new approach to conformant planning using classical planners. We apply the method on several benchmark and the result shows that the method is applicable in most of these domains.

CPCL—A New Approach to Conformant Planning

We now describe a new approach, called CPCL, which solves a conformant planning problem by solving several classical planning problems.

In general, a conformant planning problem can be solved in the same manner of FF-Replan by (*i*) pick an arbitrary state in the initial set of states (*ii*) computing a new solution for the classical problem of this state, which we often call sub-problem, using a classical planner (FF), and (*iii*) checking if the solution is also the plan for the conformant planning problem, if yes return the solution, otherwise go back to step (*ii*).

Even though this process is theoretically sound, such a brute-force computation may not be practical for different reasons. First, the set of solutions of the classical planning problem is generally infinite and thus generating all solutions is impractical. Second, for efficiency and space reasons, most state-of-the-art planners use heuristics and remove some parts of the search space (non-optimal planners avoid exploring the same state twice while optimal planners ignore paths which violate some criteria, e.g., cost of current path is greater than an established threshold). Third, the process ignores the relationships among the sub-problems which are often useful in solving the problem. Hence, we need to develop an algorithm for conformant planning using a modification of the steps (**ii**)-(**iii**).

Let us motivate the types of modifications that need to be done through a simple example.

Example 1. *Consider the problem* $P = \langle \{p, q, r\}, O, I, r \rangle$ *where* O *contains* $a : \top \rightarrow p, r$ *and* $b : \top \rightarrow q$ *with* $pre(a) = q$ *and* $pre(b) = \top$, $I = \{\texttt{one-of}(q, \neg q), \neg p, \neg r\}$. *Here,* $ext(I) = \{s_0, s_1\}$ *with* $s_0 = \{q, \neg p, \neg r\}$ *and* $s_1 = \{\neg q, \neg p, \neg r\}$. *(*$\top$ *stands for true)*

Consider $P(s_0)$. *We have that* $\alpha = [a]$ *is a solution for* $P(s_0)$. α *is, however, not a solution for* $P(s_1)$ *and thus it is also not a solution for* P. *It is easy to see that if we* insert *the action* b *in front of* a*, we obtain a plan* $[b, a]$ *for* $P(s_1)$

which is also solution for P.

The example shows that a problem P can be solved by generating a solution α for one of its sub-problems and trying to create a solution of P, using α as the seed, by inserting actions to α to maintain the executability of actions in α or to achieve the goal.

Algorithm

Alg. 1 shows the main search algorithm of the planner CPCL. $plan(X)$ plays the role of a classical planner that returns a set of solutions of X. We assume that $plan(X)$ returns one solution at a time, **nil** if there is no more solution, or **failed** if X does not have a solution. $is\_solution(\beta, P)$ checks whether or not β is a solution of the problem P.

Given a problem P, CPCL randomly selects a sub-problem $P(s_0)$ of P and explores the solutions of $P(s_0)$ generated by $plan(P(s_0))$ (loop: line 6–11). In each iteration, it checks whether α_{s_0}, a plan for $P(s_0)$, is a solution of P (line 7) or an attempt is made to "repair" it so that it becomes a plan for other sub-problems $P(s_i)$ (line 8).

The procedure $completion(\alpha, P, \Sigma, Index)$ (line 8) constructs a (possible) solution of P from the sequence α. Basically, the procedure inserts actions into the sequence of actions to maintain the executability of actions in the sequence and the goal. This procedure is still in early design since there are many options for maintenance: (1) maintain only the precondition of actions (2) maintain all effects of actions or (3) a compromise between the first two options.

If the repair fails, another plan for $P(s_0)$ is generated and the process repeated. The algorithm returns either a solution of P, **failed** to indicate that P is not solvable, or **unknown** to indicate that it cannot solve the problem.

Algorithm 1 CPCL(P)

```
1:  Input: A planning problem $P = \langle F, O, I, G \rangle$
2:  Output: A solution for $P$
3:  Let $\Sigma = [s_0, \ldots, s_n] = ext(I)$        {Compute $ext(I)$}
4:  $\alpha_{s_0} = plan(P(s_0))$            {Get a solution of $P(s_0)$}
5:  if $\alpha_{s_0}$ = failed then return failed
6:  while $\alpha_{s_0} \neq$ nil do
7:      if $is\_solution(\alpha_{s_0}, P)$ then return $\alpha_{s_0}$
8:      else $\beta = completion(\alpha_{s_0}, P, \Sigma, 1)$
9:          if $is\_solution(\beta, P)$ then return $\beta$
10:     $\alpha_{s_0} = plan(P(s_0))$
11: end while
12: return unknown
```

Preliminary result and Future work

We have developed a planner, called CPCL, and tested it against state-of-the-art planners using the benchmarks from the International Planning Competition 5 and the International Planning Competition 6. The results show that the approach is applicable in 9 out of 10 domains and CPCL can solve most problems whenever the technique can be applied as can be observed in Table 1. We observe that when an appropriate strategy is used, CPCL outperforms other planners impressively. However, the test also reveals that if inappropriate strategy is used, the performance of CPCL decreases drastically due to memory consumption. Therefore, developing a strategy that is suitable for all the domains is one of challenging tasks for my thesis.

Domain(#Instances)	CPA(H)	t0	DNF	CPCL
blw(4)	3	3	3	**4**
coins(30)	20	20	20	**30**
comm(25)	**25**	**25**	**25**	**25**
sortnet(15)	**15**	9	**15**	**15**
uts(30)	30	30	30	**30**
uts-cycle(30)	11	7	12	**15**
raos-keys(30)	2	2	2	**3**
forest(9)	2	8	2	**9**
dispose(90)	66	62	89	**90**
Total (263)	174	166	198	**221**

Table 1: Number of solved problems

In the future we would like to:

1. Develop adaptable strategies for action maintenance as described in the algorithm section.
2. Investigate the affect of the selection of $s0$ on the algorithm and develop the heuristic to pick $s0$. Investigate how different orders of states in the initial belief state affect the algorithm and develop a sorting algorithm on the set of initial states.
3. Characterize conformant planning domains that the method can be applied.
4. Extend the algorithm to handle non-deterministic actions.
5. Adopt the method for contigent planning

References

D. Bryce, S. Kambhampati, and D. Smith. Planning Graph Heuristics for Belief Space Search. *JAIR*, 26:35–99, 2006.

A. Cimatti, M. Roveri, and P. Bertoli. Conformant Planning via Symbolic Model Checking and Heuristic Search. *AIJ*, 159:127–206, 2004.

H. Palacios and H. Geffner. Compiling Uncertainty Away: Solving Conformant Planning Problems Using a Classical Planner (Sometimes). In *AAAI*, 2006.

H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *JAIR*, 35:623–675, 2009.

D.E. Smith and D.S. Weld. Conformant graphplan. In *AAAI*, pages 889–896, 1998.

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

S. T. To, T. C. Son, E. Pontelli. A New Approach to Conformant Planning using CNF. *ICAPS 2010.*

D. V. Tran, H. K. Nguyen, E. Pontelli, and T. C. Son. Improving performance of conformant planners: Static analysis of declarative planning domain specifications. In *PADL 2009*, pages 239–253. Springer, 2009.

S.W. Yoon, A. Fern, and R. Givan. FF-Replan: A baseline for probabilistic planning. *ICAPS*, 352–259. AAAI. 2007.