Conformant Planners: Approximation vs. Representation

Son Thanh To, Dang-Vien Tran, Hoang-Khoi Nguyen, Tran Cao Son, Enrico Pontelli
Knowledge representation, Logic, and Advanced Programming Laboratory
Computer Science Department
New Mexico State University
Las Cruces, NM 88003

Abstract

This demonstration presents three conformant planners: CPA, DNF, and CNF, whose performances are comparable to those of state-of-the-art conformant planners. All three are best-first search and progression-based planners. CPA, a representative of approximation-based planners, was the recipient of the *Best Non-Observable Non-Deterministic Planner Award* at IPC-2008. DNF and CNF are complete planners which employ different representations for belief states and perform better than CPA in several domains. DNF uses DNF-formulae which are minimal with respect to set inclusion, while CNF uses CNF-formulae which are minimal with respect to subsumption. The key difference between these two planners and CPA lies in that they implement an algorithm for guaranteeing completeness of the planner only when needed, while CPA does so before the search for a plan starts. The heuristics employed by the three aforementioned planners are combinations of two well-know heuristics used in conformant planning: the size of the belief state and the number of satisfied subgoals. The demonstration also presents various techniques that contribute to the performance and scalability of these planners. Lessons learned during the development of these planners are discussed.

Introduction

Conformant planners deal with planning problems with uncertainty about the initial states. The following issues are key to the development of a conformant planner:

- A formalization of actions in presence of incomplete information; and
- A belief state representation and a good heuristic function.

The first item is important for the correctness of the planner, as it provides the theoretical foundations for the planner to progress in the presence of incomplete information. The second item is critical to the performance and scalability of the planner, since the complexity of the problem of computing the successor belief state is, in general, computationally expensive.

In this demonstration, we introduce three conformant planners, CPA, DNF, and CNF. These planners employ different approaches to searching for a solutions. CPA searches for a solution in the space of sets of partial states instead of the space of belief states (Son et al. 2005). CPA uses DNF-formulae to represent a set of partial states and employs several techniques to reduce the size of the initial belief state. CPA was the recipient of the *Best Non-Observable Non-Deterministic Planner Award* at IPC-2008. The exceptional performance of CPA led us to study the impact of the belief state representation on the performance of complete conformant planners. This investigation resulted in the development of two planners, DNF and CNF, which use DNF- and CNF-formulae, respectively, in encoding belief states.

We begin with a short review of the background of conformant planning. We then discuss the basic concepts used in the development of the planners. Afterwards, we describe the organization of the systems.

Background: Conformant Planning

A *planning problem* is described by a tuple $P = \langle F, O, I, G\rangle$, where F is a set of propositions, O is a set of actions, I describes the initial state of the world, and G describes the goal. A *literal* is either a proposition $p \in F$ or its negation $\neg p$. $\bar{\ell}$ denotes the complement of a literal ℓ—i.e., $\bar{\ell} = \neg\ell$, where $\neg\neg p = p$ for $p \in F$. For a set of literals L, $\overline{L} = \{\bar{\ell} \mid \ell \in L\}$. A conjunction of literals is often viewed as the set of its literals.

A set of literals X is *consistent* if there exists no $p \in F$ such that $\{p, \neg p\} \subseteq X$. A set of literals X is *complete* if for each $p \in F$, $\{p, \neg p\} \cap X \neq \emptyset$. A *state* s is a consistent and *complete* set of literals. A *belief state* is a set of states.

Each action a in O is associated with a precondition ϕ (denoted by $pre(a)$) and a set of conditional effects of the form $\psi \rightarrow \ell$ (also denoted by $a : \psi \rightarrow \ell$), where ϕ and ψ are sets of literals and ℓ is a literal.

A state s satisfies a literal ℓ, denoted by $s \models \ell$, if $\ell \in s$. s satisfies a conjunction of literals X, denoted by $s \models X$, if it satisfies every literal belonging to X. The satisfaction of a formula in a state is defined in the usual way. Likewise, a belief state S satisfies a literal ℓ, denoted by $S \models \ell$, if $s \models \ell$ for every $s \in S$. S satisfies a conjunction of literals X, denoted by $S \models X$, if $s \models X$ for every $s \in S$.

Given a state s, an action a is *executable* in s if $s \models pre(a)$. The effect of executing a in s is

$$e(a, s) = \{\ell \mid \exists(a : \psi \rightarrow \ell).\ s \models \psi\}$$

The transition function, denoted by Φ, in the planning do-

main of P is defined by

$$\Phi(a,s) = \begin{cases} s \setminus \overline{e(a,s)} \cup e(a,s) & s \models pre(a) \\ \bot & \text{otherwise} \end{cases} \quad (1)$$

where $\bot$ denotes a fail state.

Φ is extended to define $\widehat{\Phi}$, a transition function which maps sequences of actions and belief states to belief states for reasoning about the effects of plans. Let S be a belief state. We say that an action a is executable in a belief state S if it is executable in every state belonging to S. Let $\alpha_n = [a_1, \ldots, a_n]$ be a sequence of actions and $\alpha_i = [a_1, \ldots, a_i]$:

- If $n = 0$ then $\widehat{\Phi}([\,], S) = S$;
- If $n > 0$ then
  - if $\widehat{\Phi}(\alpha_{n-1}, S) = \bot$ or a_n is not executable in $\widehat{\Phi}(\alpha_{n-1}, S)$, then $\widehat{\Phi}(\alpha_n, S) = \bot$;
  - if $\widehat{\Phi}(\alpha_{n-1}, S) \neq \bot$ and a_n is executable in $\widehat{\Phi}(\alpha_{n-1}, S)$ then $\widehat{\Phi}(\alpha_n, S) = \{\Phi(a_n, s') \mid s' \in \widehat{\Phi}(\alpha_{n-1}, S)\}$.

The initial state of the world I is a belief state and is represented by a formula. In our investigation, we consider I to be a conjunction of literals, `oneof` statements and `or` statements—where a `oneof` statement (`or` statement) represents an exclusive-or (resp. logical or) of its components. By S_I we denote the set of all states satisfying I. Typically, the goal description G can contain literals and `or` statements.

A sequence of actions $\alpha = [a_1, \ldots, a_n]$ is a solution of P if $\widehat{\Phi}(\alpha, S_I)$ satisfies G. In this paper, we will denote with C_a the set of conditional effects of an action a.

Approximation-Based Planning

The approach to approximation-based planning adopted in CPA relies on the 0-approximation semantics for reasoning about effects of actions in presence of incomplete information about the initial state (Son and Baral 2001). Intuitively, the approach (*i*) replaces a belief state by a *partial state*, which is a set of fluent literals; and (*ii*) specifies how to compute the successor partial state, i.e., the result of executing an action in a given partial state. This is appealing for conformant planning, since it lowers the complexity of conformant planning (Baral, Kreinovich, and Trejo 2000). It is characterized by a function (Φ_A) that maps an action and a partial state to a partial state. Given a partial state δ, the possible effects of a in δ are given by

$$pc_a(\delta) = \{l \mid (\psi \rightarrow l) \in C_a, \overline{\psi} \cap \delta = \emptyset\}. \quad (2)$$

The successor partial state from the execution of a in δ is defined by $\Phi_A(a,\delta) = (\delta \cup e(a,\delta)) \setminus \overline{pc_a(\delta)}$ if a is executable in δ; and $\Phi_A(a,\delta) = \bot$, otherwise. This function is then extended to define $\widehat{\Phi_A}$, similarly to $\hat{\Phi}$, to reason about plans.

Observe that Φ_A coincides with Φ under complete information, i.e., $\Phi_A(a,s) = \Phi(a,s)$ for every state s. However, Φ_A can be incomplete. For example, given a planning problem P_1 with the set of propositions $\{f, g, h\}$, the initial state $I = \emptyset$, the set of actions $O = \{a : f \wedge g \rightarrow h, a : f \wedge \neg g \rightarrow h, b : g \rightarrow f, b : \neg g \rightarrow f\}$, and the goal $G = \{h\}$. $\Phi_A(b, \{\emptyset\}) = \{\emptyset\}$, i.e., Φ_A will answer the query whether f will be true after the execution of b in the initial state with 'No' whereas Φ will say 'Yes.'

To guarantee completeness, CPA exploits the completeness condition in (Son and Tu 2006) to identify a minimal set of initial partial states and searches for solutions in the space of sets of partial states, called *cs-states*.

DNF and CNF

The DNF planner uses DNF-formulae to represent belief states and employs the transition function Φ_{DNF} in its progression. Φ_{DNF} relies on an algorithm for splitting a partial state δ into a *minimal* set of partial states Δ (with respect to set inclusions among members) such that every precondition of a given action a is either true or false in each member of Δ. The details of Φ_{DNF} can be found in (To, Pontelli, and Son 2009).

For example,given the planning problem P_1 described in the previous subsection, the behavior of Φ_{DNF} is as follows. The initial belief state will be represented by the DNF-formula $\Delta = \{\emptyset\}$. $\Phi_{DNF}(b, \Delta)$ will begin with the realization that Δ will need to be split into $\Delta_1 = \{\{g\}, \{\neg g\}\}$. The result of executing b in Δ will then be $\Delta_2 = \{\{f, g\}, \{f, \neg g\}\}$. Executing a in Δ_2 does not require any splitting and results in $\Delta_3 = \{\{f, g, h\}, \{f, \neg g, h\}\}$.

Observe that the initial belief state consists of eight states (all possible states of the problem) and the DNF-formula after the splitting for b contains only two elements. This representation allows for an efficient computation of the successor belief state—i.e., $\Phi_{DNF}(a, \Delta)$ can be computed in polynomial time in the size of Δ, under a reasonable assumption that the number of effects of each action is bounded, for every action a and DNF-formula Δ. It is worth to mention that the belief state representation of DNF is similar to that of CPA. In this sense, the key distinction between DNF and CPA lies in that DNF computes the set of partial states needed for guaranteeing the completeness of the planner only when needed, while CPA computes it before the search process starts. Both planners, however, still suffer from the possible huge size of the initial state.

To address the problem of the size of the initial state faced by DNF and CPA, the CNF planner uses CNF-formulae, represented as a set of clauses and *minimal* with respect to subsumption and unit propagation, to represent belief states. The transition function Φ_{CNF} of CNF is similar to Φ_{DNF}, in that it also relies on an algorithm for splitting a formula φ into a set of CNF-formulae, $enb(a, \varphi)$, such that the effects of the action a on φ can be determined. For example, Φ_{CNF} behaves almost as Φ_{DNF} with respect to the problem P_1, as the initial state $\emptyset$ is splitted into two clauses $\{g\}$ and $\{\neg g\}$, enabling the execution of b to achieve f from the initial state. In this sense, the key difference between DNF and CNF lies in their use of different belief state representations. While the computation of Φ_{CNF} is more complex than that of Φ_{DNF}, its representation of the initial state is more compact. This representation pays off when the size of the initial state is huge. This representation also allows for the application of a new technique, called relaxation of `oneof` statements, which allows CNF to solve all instances of the `coin`-domain. To the best of our knowledge, CNF

is the only planner can deal with the instances `coins-21` to `coins-30`, whose initial belief states contain more than 10^6 states. Precise definition of Φ_{CNF} can be found in (To, Son, and Pontelli 2010).

Analysis and Simplifications

The analysis and simplification techniques implemented in the three planners help simplify the planning instances by reducing the number of actions and propositions. It also contains, for each representation, a technique that reduces the size of the initial cs-state (or belief state). These techniques briefly discussed next.

Basic Simplifications: We consider two well-known basic steps: *forward reachability* and *goal relevance*. Several planners implement these two steps.

Forward reachability is used to detect: *(i)* propositions whose truth value cannot be affected by the actions in the problem specification (w.r.t. the initial state); *(ii)* actions whose execution cannot be triggered w.r.t. the given initial state. This process can be modeled as a fixpoint computation. Goal relevance proceeds in a similar manner, by detecting actions that are relevant to the achievement of the goal.

Combination of `oneof` *Statements:* `oneof` statements are used to specify the uncertainty about some propositions and/or mutual exclusion between propositions. The number of the `oneof` statements and their size (the size of an `oneof` statements is the number of its elements) determine the size of the initial cs-state.

The idea of the combination of `oneof` statements technique is based on the *non*-interaction between actions and propositions in different sub-problems of a conformant planning problem. This idea is best illustrated with a simple example.

Let us consider the planning problem P_2 with the set of propositions $\{f, g, h, p, i, j\}$, the initial state $I = \{\texttt{oneof}(f, g), \texttt{oneof}(h, p), \neg i, \neg j\}$, the set of actions $O = \{\ a : f \rightarrow i \quad c : h \rightarrow j \quad b : g \rightarrow i \quad d : p \rightarrow j\ \}$, and the goal $G = i \wedge j$. Here, a causes i to be true if f is true; c causes j to be true if h is true; b causes i to be true if g is true; and d causes j to be true if p is true.

It is easy to see that the sequence $\alpha = [a, b, c, d]$ is a solution of P_2. Furthermore, the search should start from the cs-state consisting of the four states:

$$\{f, \neg g, h, \neg p, \neg i, \neg j\} \quad \{\neg f, g, h, \neg p, \neg i, \neg j\}$$
$$\{f, \neg g, \neg h, p, \neg i, \neg j\} \quad \{\neg f, g, \neg h, p, \neg i, \neg j\}$$

Let P_2' be the problem obtained from P_2 by replacing I with I', where $I' = \{\texttt{oneof}(f \wedge h, g \wedge p), \neg i, \neg j\}$.

We can see that α is also a solution of P_2'. Furthermore, each solution of P_2' is a solution of P_2. This transformation is interesting since the initial cs-state now consists only of two states: $\{f, \neg g, h, \neg p, \neg i, \neg j\}$ and $\{\neg f, g, \neg h, p, \neg i, \neg j\}$. In other words, the number of states in the initial belief state (or initial cs-state) that a conformant planner has to consider in P_2' is 2, while it is 4 in P_2. This transformation is possible because the set of actions that are "activated" by f and g is disjoint from the set of actions that are "activated" by h and p, i.e., $preact(\{f, g\}) \cap preact(\{h, p\}) = \emptyset$ where $preact(\delta)$ we denote the set of actions depending on δ.

Using this technique, many `oneof` statements can be combined into one, yielding several order of magnitudes reduction in the size of the initial cs-state.

Goal Splitting: The key idea is that if a problem P contains a subgoal whose truth value cannot be negated by the actions used to reach the other goals, then the problem can be decomposed into smaller problems with different goals, whose solutions can be combined to create a solution of the original problem. This technique can be seen as a variation of the goal ordering technique in (Hoffmann, Porteous, and Sebastia 2004) and relies on the notion of dependence proposed in (Son and Tu 2006).

Relaxation of `oneof` *Statements:* In contrast to the combination of `oneof` statements technique, a relaxation of an `oneof` statement increases the number of the states in the initial belief state. More precisely, the relaxation of a `oneof` statement $\texttt{oneof}(l_1, \ldots, l_k)$ replaces it with an `or` statement $\texttt{or}(l_1, \ldots, l_k)$. Conditions for the soundness of the transformation have been identified. This technique also relies on the non-interaction between actions and propositions in these `oneof` statements. Let us illustrate this with a simple example.

Let consider the planning problem P_3 with the set of propositions $\{f, g, i, j\}$, the initial state $I = \{\texttt{oneof}(f, g), \neg i, \neg j\}$, the set of actions $O = \{\ a : f \wedge \neg i \rightarrow i, b : g \wedge \neg j \rightarrow j\ \}$, and the goal $G = i \wedge j$. Any solution of the problem P_3' with the same set of propositions, the initial state $I' = \{\texttt{or}(f, g), \neg i, \neg j\}$, O, and G will be a solution of P_3.

Observe that this technique increases the number of states in the initial belief state. However, the size of the CNF formula representing the relaxation will be smaller compared to the size of the CNF formula representing the original belief state.

Heuristics in CPA

The heuristics used in CPA are the combination of the following well-known heuristics.

- *The cardinality heuristic:* we prefer cs-states that have a smaller cardinality. In other words, $h_{card}(\Sigma) = |\Sigma|$ where Σ is a cs-state. Note that we use this heuristic in a forward fashion, and hence, is different from its use in (Bertoli, Cimatti, and Roveri 2001; Bryce and Kambhampati 2004). The intuition behinds this is that planning with complete information is "easier" than planning with incomplete information, and a lower cardinality implies a lower degree of uncertainty.
- *The number of satisfied subgoals:* denoted by $h_{goal}(\Sigma)$.

CPA uses the combination: $h_{cs}(\Sigma) = (h_{card}(\Sigma), h_{goal}(\Sigma))$ with lexicographic ordering. It gives preference to the cs-states with a lower degree of uncertainty, i.e., cs-states that have a smaller cardinality. If the cardinality of two cs-states does not differ, then the heuristics gives preference to those cs-states that maximize the number of satisfied subgoal.

Heuristics in DNF and CNF

Given a DNF-state Δ, the heuristic function used in DNF is a combination of the following three values (along with a lexicographic ordering):

- $h_{goal}(\Delta)$: the number of subgoals satisfied by Δ.
- $h_{card}(\Delta)$: the cardinality of Δ.
- $h_{dis}(\Delta)$: the *square distance* of Δ to the goal, defined by

$$h_{dis}(\Delta) = \Sigma_{\delta \in \Delta}(|G| - h_{goal}(\delta))^2$$

where G is the goal of the problem.

For a CNF-state φ, encoded by a set of clauses, the heuristic function of CNF is a combination of the number of satisfied subgoals in φ ($h_{goal}(\varphi)$) and the size of φ, defined as the sum of the sizes of non-unit clauses in φ ($h_{size}(\varphi)$).

System Organization

The proposed systems are organized as in Fig. 1. The first component is a front-end, that acts as a *static analyzer*. The static analyzer is in charge of applying several simplifications and optimizations to the input problem specification—initially expressed in PDDL. The simplified specification (expressed either in PDDL or in the action language $\mathcal{AL}$—the native input format of CPA, DNF, and CNF) produced by the static analyzer is then fed to the actual planner. The separation of the two stages allows us to investigate the use of different planners applied to the same simplified problem specification.

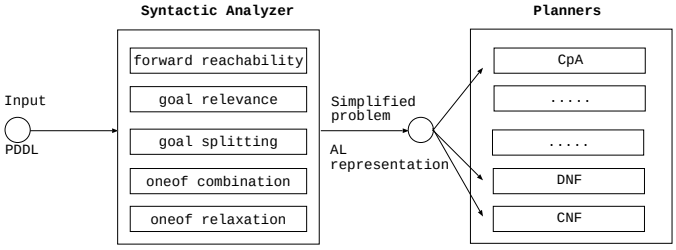

Figure 1: Overall System

The implementation of the static analyzer makes use of the PDDL parser originally developed for these systems; the parser has been modified to enable the construction of a Prolog representation of the problem specification. This Prolog representation is used as the input to the static analyzer, implemented in Prolog. The analyzer implements the basic simplifications, the `oneof`-combination/relaxation, and the goal-splitting algorithm. Its output is a sequence of simplified problems in $\mathcal{AL}$, which serve as input to these planners. An option is also available to produce PDDL output from the static analyzer—that can be fed, for example, to a different planner.

CPA makes use of h_{cs} in combination with a best-first search algorithm. CPA employs an explicit representation of cs-states as sets of sets of propositions, and they make use of the C++ standard library `std` for sets manipulation. To reduce the space consumption, a partial state is created only once and it is shared by all cs-states containing it.

Discussion and Conclusion

We presented the main techniques implemented in the three conformant planners CPA, DNF, and CNF. Experimentally, these planners are competitive with state-of-the-art conformant planners in several benchmark domains. CNF scales better than others in some benchmarks but the overhead in computing the successor belief state slows it down in small instances.

The development of these planners highlights the fact that the representation of belief states can significantly impact the performance of a planner. Some simplification techniques are applicable to a wide range of representations, while others are specific to certain representations. For example, `oneof`-combination is better for DNF-representation and the `oneof`-relaxation is better for CNF-representation.

So far, the proposed planners do focus only on the size of the initial belief state. For scalability, the problems related to the number of actions in the planning problems will need to be addressed as well. We believe that heuristics should provide a solution to this problem and this will be a focus of our future work. Furthermore, our study reveals that there seems to be no "one size fits all" representation for all planning domains. As such, identifying the most useful representation given a planning problem will be another interesting work that we plan to explore in the near future.

References

Baral, C.; Kreinovich, V.; and Trejo, R. 2000. Computational complexity of planning and approximate planning in the presence of incompleteness. *AIJ* 122:241–267.

Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Heuristic search + symbolic model checking = efficient conformant planning. In *IJCAI*, 467–472. Morgan Kaufmann.

Bryce, D., and Kambhampati, S. 2004. Heuristic Guidance Measures for Conformant Planning. In *ICAPS 2004*, 365–375. AAAI.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res.* 22:215–278.

Son, T. C., and Baral, C. 2001. Formalizing sensing actions - a transition function based approach. *AIJ* 125(1-2):19–91.

Son, T. C., and Tu, P. H. 2006. On the Completeness of Approximation Based Reasoning and Planning in Action Theories with Incomplete Information. In *KR*, 481–491.

Son, T. C.; Tu, P. H.; Gelfond, M.; and Morales, R. 2005. Conformant Planning for Domains with Constraints — A New Approach. In *AAAI*, 1211–1216.

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

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