Conjunctive Representations in Contingent Planning: Prime Implicates versus Minimal CNF Formula[*]

Son Thanh To and **Tran Cao Son** and **Enrico Pontelli**
New Mexico State University
Department of Computer Science
sto|tson|epontell@cs.nmsu.edu

Abstract

This paper compares in depth the effectiveness of two conjunctive belief state representations in contingent planning: *prime implicates* and *minimal CNF*, a compact form of CNF formulae, which were initially proposed in conformant planning research (To *et al.* 2010a; 2010b). Similar to the development of the contingent planner CNF_{ct} for minimal CNF (To *et al.* 2011b), the present paper extends the progression function for the prime implicate representation in (To *et al.* 2010b) for computing successor belief states in the presence of incomplete information to handle non-deterministic and sensing actions required in contingent planning. The idea was instantiated in a new contingent planner, called PI_{ct}, using the same AND/OR search algorithm and heuristic function as those for CNF_{ct}. The experiments show that, like CNF_{ct}, PI_{ct} performs very well in a wide range of benchmarks. The study investigates the advantages and disadvantages of the two planners and identifies the properties of each representation method that affect the performance.

Introduction

Contingent planning (Peot and Smith 1992) has been considered one of the most challenging problems in automated planning (Haslum and Jonsson 1999). It is charged with finding a plan for an agent to achieve the goal in the presence of imperfect knowledge about the world, non-deterministic actions, and observations (also called sensing actions).

Most state-of-the-art contingent planners, e.g., MBP (Bertoli et al. 2001), POND (Bryce *et al.* 2006), and contingent-FF (Hoffmann and Brafman 2005), employ an AND/OR search algorithm in the belief state space for contingent solutions. Those planners have been shown to be fairly effective in a number of domains. Yet, their scalability is still modest, mostly due to the disadvantages of the method they use to represent belief states as discussed in (To *et al.* 2011a; 2011b). In those papers, we also discuss the advantage and weakness of the translation-based approach introduced in (Albore, Palacios, and Geffner 2009).

Recently, in (To *et al.* 2011a), we proposed a new approach to contingent planning which relies on the DNF representation of belief states for conformant planning (To *et al.* 2009). We extended the progression function for DNF to handle non-deterministic and sensing actions and developed a new AND/OR forward search algorithm, called PrAO, for contingent planning. The resulting planner, called DNF_{ct}, outperforms other state-of-the-art contingent planners by far on most benchmarks. Nevertheless, DNF_{ct} does not perform well in problems where the size of disjunctive formulae representing the belief states is too large. This issue motivated us to develop CNF_{ct} (To *et al.* 2011b) that employs the same search algorithm PrAO as DNF_{ct} but uses CNF for belief state representation. A comparison on the effectiveness of the two representations, DNF and CNF, is presented in (To *et al.* 2011b). That study shows that neither of the two methods completely dominates the other. While DNF_{ct} can find a solution faster on more problems, CNF_{ct} scales better with the size of problems, especially those where the size of disjunctive formulae representing the belief states is significantly larger. The main reason for this lies in that the trade-off between the complexity for successor belief state computation and the size of formulae representing belief states, which affects also the computation and scalability.

The aforementioned study motivates us to search for a middle ground between DNF and CNF representations of belief states. Ideally, we would like to take the advantages of both representations. This inspires us to investigate the use of *prime implicates* (*pi-formula*) to represent belief states. In this work, we extend the progression function defined in (To *et al.* 2010b) to handle non-deterministic and sensing actions required for contingent planning and implements it in a contingent planner, called PI_{ct}, that uses the same search algorithm and same heuristic function as CNF_{ct} and DNF_{ct} in (To *et al.* 2011b) do. The experiments validate our expectation: PI_{ct} is faster than CNF_{ct}, scales better than DNF_{ct}, and outperforms the other state-of-the-art planners on most tested domains. The paper then compares the advantages and disadvantages of pi-formula and CNF, identifies properties of the representation schemes that affect the performance of the planners differently over the benchmarks.

The rest of the paper is organized as follows. The next section reviews the basics of contingent planning. Afterwards, the prime implicate representation and CNF representation in (To *et al.* 2010b) are extended for contingent planning. It follows with a description of PI_{ct} and CNF_{ct}, an empirical evaluation against state-of-the-art contingent plan-

[*]The authors are partially supported by NSF grant IIS-0812267.

ners. Next, a discussion about the effectiveness of the representations is presented. The paper ends with a summary of the contribution and the future work.

Background: Contingent Planning

A *contingent planning problem* is a tuple $P = \langle F, A, \Omega, I, G\rangle$, where F is a set of propositions, A is a set of actions, Ω is a set of observations, I describes the initial state, and G describes the goal. A and Ω are disjoint, i.e., $A \cap \Omega = \emptyset$. 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 $\overline{\neg p} = p$ for $p{\in}F$. For a set of literals L, $\overline{L} = \{\bar{\ell} \mid \ell \in L\}$. We will often use a set of conjuncts to represent a conjunction of literals.

A set of literals X is *consistent* (resp. *complete*) if for every $p{\in}F$, $\{p, \neg p\} \not\subseteq X$ (resp. $\{p, \neg p\} \cap X \neq \emptyset$). A *state* is a consistent and complete set of literals. A *belief state* is a set of states. We will often use lowercase (resp. uppercase) letters to represent a state (resp. a belief state).

Each action a in A is a tuple $\langle pre(a), O(a)\rangle$, where $pre(a)$ is a set of literals indicating the preconditions of action a and $O(a)$ is a set of action outcomes. Each $o(a)$ in $O(a)$ is a set of conditional effects $\psi \rightarrow \ell$ (also written as $o_i : \psi \rightarrow \ell$), where ψ is a set of literals and ℓ is a literal. If $|O(a)| > 1$ then a is non-deterministic. $O(a)$ is mutually exclusive, i.e., the execution of a makes one and only one outcome in $O(a)$ occur. However, which outcome that occurs is uncertain. Each observation ω in Ω is a tuple $\langle pre(\omega), \ell(\omega)\rangle$, where $pre(\omega)$ is the preconditions of ω (a set of literals) and $\ell(\omega)$ is a literal.

A state s satisfies a literal ℓ ($s \models \ell$) if $\ell \in s$. s satisfies a conjunction of literals X ($s \models X$) if $X \subseteq s$. 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 w.r.t. an outcome o_i is

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

Let $res(o_i, s) = s \setminus \overline{e(o_i, s)} \cup e(o_i, s)$. The progression function maps an action and a belief state to a belief state, defined as $\Phi(a, S) = \{res(o_i, s) \mid s \in S, o_i \in O(a)\}$ if $S \neq \emptyset$ and $S \models pre(a)$; $\Phi(a, S) = \mathit{undefined}$, otherwise.

Example 1. Given a domain $F = \{at(p_1), at(p_2), at(p_3)\}$, a belief state S that contains only one state $s = \{at(p_1), \neg at(p_2), \neg at(p_3)\}$, an action $leave(p_1)$ with $pre(leave(p_1)) = \{at(p_1)\}$ and $O(leave(p_1)) = \{o_1, o_2\}$, where $o_1 = \{\emptyset \rightarrow \neg at(p_1), \emptyset \rightarrow at(p_2)\}$ and $o_2 = \{\emptyset \rightarrow \neg at(p_1), \emptyset \rightarrow at(p_3)\}$. One can easily compute: $res(o_1, s) = \{\neg at(p_1), at(p_2), \neg at(p_3)\}$, and $res(o_2, s) = \{\neg at(p_1), \neg at(p_2), at(p_3)\}$. Hence, $\Phi(leave(p_1), S) = \{\{\neg at(p_1), at(p_2), \neg at(p_3)\}, \{\neg at(p_1), \neg at(p_2), at(p_3)\}\}$.

Observe that the non-deterministic action $leave(p_1)$ causes the certain belief state S to become uncertain.

Let ω be an observation in Ω; we define $S^+_\omega = \{s \mid s \in S, s \models \ell(\omega)\}$ and $S^-_\omega = \{s \mid s \in S, s \models \overline{\ell(\omega)}\}$.

Given a contingent planning problem P, a structure T constructed from the actions and observations of P is said to be a *transition tree* of P if

- T is empty, denoted by $[]$, or $T = a$, where $a \in A$; or
- $T = a \circ T'$, where $a \in A$ and T' is a non-empty transition tree; or
- $T = \omega(T^+|T^-)$, where $\omega \in \Omega$ and T^+ and T^- are transition trees.

Intuitively, a transition tree represents a conditional plan, as defined in the literature, and can be represented as an AND/OR graph whose nodes are belief states and links are AND/OR-edges. Let $\perp$ denote $\mathit{undefined}$. The result of the execution of a transition tree T in a belief state S, denoted by $\widehat{\Phi}(T, S)$, is a set of belief states defined as follows:

- If $S = \perp$ or $S = \emptyset \wedge T \neq []$ then $\widehat{\Phi}(T, S) = \perp$; else
- If $S \neq \emptyset$ then $\widehat{\Phi}([], S) = \{S\}$, else $\widehat{\Phi}([], \emptyset){=}\emptyset$; else
- If $T = a$, $a \in A$, then $\widehat{\Phi}(a, S) = \{\Phi(a, S)\}$; else
- If $T = a \circ T'$, $a{\in}A$, then $\widehat{\Phi}(T, S) = \widehat{\Phi}(T', \Phi(a, S))$; else
- If $T = \omega(T^+|T^-)$ then $\widehat{\Phi}(T, S) = \widehat{\Phi}(T^+, S^+_\omega) \cup \widehat{\Phi}(T^-, S^-_\omega)$ if $S \models pre(\omega)$, and $\widehat{\Phi}(T, S) = \perp$ otherwise.

Note that the definition of $\widehat{\Phi}$ allows the application of an observation ω in a belief state where $\ell(\omega)$ is known, if the subtree rooted at the resulting empty belief state is empty.

Let S_I be the initial belief state described by I. A transition tree T is said to be a solution of P if $\widehat{\Phi}(T, S_I) \neq \{\emptyset\}$ and every belief state in $\widehat{\Phi}(T, S_I) \setminus \{\emptyset\}$ satisfies the goal G.

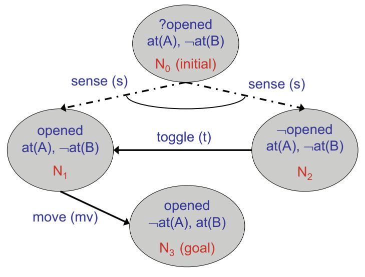

Figure 1: A contingent solution in AND/OR forward search

Example 2 ((To *et al.* 2011b)). Consider a robot R which needs to move (*mv*) from room A to room B through a door D whose opened-closed state is unknown. R can sense (*s*) whether D is opened and toggle (*t*) to change its status. The planning problem P is specified by $F{=}\{at(A), at(B), opened\}$, $A = \{mv, op\}$, $\Omega = \{s\}$, $I = at(A)$, and $G = at(B)$; where $mv = (opened, \{at(A){\rightarrow}\neg at(A), at(A){\rightarrow}at(B), at(B){\rightarrow}\neg at(B), at(B){\rightarrow}at(A)\})$, $t = (true, \{\neg opened{\rightarrow}opened, opened{\rightarrow}\neg opened\})$, and $\ell(s) = opened$.

Figure 1 illustrates a solution for the problem, which is rather a directed acyclic graph (DAG) than a tree and corresponds to the transition tree $s(mv \mid t \circ mv)$. The solution depth, which is the length of the longest action sequence that leads to a goal node from the initial node ($s \circ t \circ mv$), is 3.

The next section extends the progression functions in (To *et al.* 2010b) to deal with non-deterministic actions and observations for contingent planning.

Prime Implicates and CNF Representations

A *clause* α is a disjunctive set of literals. α is *tautological* if $\{f, \neg f\} \subseteq \alpha$ for some $f \in F$. α is a *unit clause* if it contains only one literal, called a *unit literal*. A *CNF-formula* is a set of clauses. A literal ℓ is in a CNF formula φ, denoted by $\ell \in \varphi$, if there exists $\alpha \in \varphi$ such that $\ell \in \alpha$. By φ_ℓ we denote the set of clauses in φ which contain ℓ.

A clause α *subsumes* a clause β if $\alpha \subset \beta$. A clause α of a CNF-formula φ is said to be *trivially redundant* for φ if it is either tautological or subsumed by another clause in φ. The technique of simplifying a CNF formula by removing the trivially redundant clauses from it is called *reduction*.

Two clauses α and β are said to be *resolvable* if there exists a literal ℓ such that $\ell \in \alpha$, $\bar{\ell} \in \beta$, and their *resolvent* $\alpha|\beta$, defined by $\alpha|\beta = (\alpha \setminus \{\ell\}) \cup (\beta \setminus \{\bar{\ell}\})$, is a non-tautological clause. Observe that, if α and β are two resolvable clauses in a CNF formula φ then φ can be simplified to an equivalent smaller CNF formula by replacing the set of clauses subsumed by $\alpha|\beta$ in φ with $\alpha|\beta$, provided that this set is not empty (otherwise the formula will be increased by $\alpha|\beta$). This technique is referred to as *subsumable resolution*.

A clause α is said to be an *implicate* of a formula φ if $\varphi \models \alpha$. It is a *prime implicate* of φ if there is no other implicate β of φ such that β subsumes α. By $PI(.)$ we denote the function that returns the set of prime implicates of a formula. A CNF formula φ is a *prime implicate formula* (*pi-formula*) if $\varphi = PI(\varphi)$.

A *PI-state* is a pi-formula. A *PI-belief state* is a set of PI-states.

A CNF formula φ is said to be a *CNF-state* if

- φ does not contain a trivially redundant clause; and
- φ does not contain two resolvable clauses γ and δ such that $\gamma|\delta$ subsumes a clause in φ.

Observe that a CNF-state is *minimal* in the sense that it cannot be simplified to a smaller CNF-formula using reduction or subsumable resolution. Conversely, one can use these techniques to simplify an arbitrary CNF-formula to an equivalent (usually smaller) CNF-state. By $min(.)$, we denote an idempotent function that maps a CNF-formula to an equivalent CNF-state. Note that a PI-state is, by definitions, also a CNF-state but the converse is not necessarily true.

A set of CNF-states is called a *CNF-belief state*.

For two CNF formulae $\varphi = \{\alpha_1, \ldots, \alpha_n\}$ and $\psi = \{\beta_1, \ldots, \beta_m\}$, the *cross-product* of φ and ψ, denoted by $\varphi \times \psi$, is the CNF-formula defined by $\{\alpha_i \cup \beta_j \mid \alpha_i \in \varphi, \beta_j \in \psi\}$. If either φ or ψ is empty then $\varphi \times \psi = \emptyset$. The *reduced-cross-product* (resp. *min-cross-product*) of φ and ψ, denoted by $\varphi \otimes \psi$ (resp. $\varphi \otimes_{min} \psi$), is the CNF-formula obtained from $\varphi \times \psi$ by removing all trivially redundant clauses in $\varphi \times \psi$ (resp. CNF-state $min(\varphi \times \psi)$). For a set of CNF formulae $\Psi = \{\varphi_1, \ldots, \varphi_n\}$, $\times[\Psi]$ (resp. $\otimes[\Psi]$, $\otimes_{min}[\Psi]$) denotes $\varphi_1 \times \ldots \times \varphi_n$ (resp. $\varphi_1 \otimes \ldots \otimes \varphi_n$, $\varphi_1 \otimes_{min} \ldots \otimes_{min} \varphi_n$). It is easy to see that $\times[\Psi]$, $\otimes[\Psi]$, and $\otimes_{min}[\Psi]$ are a CNF-formula equivalent to $\bigvee_{i=1}^{n} \varphi_i$.

It can be proved that the reduced-cross-product of a set of pi-formulae (PI-belief state) is also a pi-formula (PI-state).

Let o_i be an outcome of action a. A CNF-formula φ is *enabling* for o_i if for every conditional effect $o_i : \psi \rightarrow \ell$, either $\varphi \models \psi$ or $\varphi \models \neg\psi$ holds. A set of CNF-formulae Ψ is *enabling* for o_i if every $\varphi \in \Psi$ is enabling for o_i.

For an enabling CNF-formula φ for o_i, the effect of a in φ when the outcome o_i occurs, denoted by $e(o_i, \varphi)$, is defined by: $e(o_i, \varphi) = \{\ell \mid o_i : \psi \rightarrow \ell, \varphi \models \psi\}$.

The update of a CNF-state φ by a literal ℓ, denoted by $upd(\varphi, \ell)$, is a CNF-state defined by:

$$upd(\varphi, \ell) = min((\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}})) \wedge \ell \wedge \varphi_\ell|\varphi_{\bar{\ell}})$$

where $\varphi_\ell|\varphi_{\bar{\ell}} = \{\alpha|\beta \mid \alpha \in \varphi_\ell, \beta \in \varphi_{\bar{\ell}}, \alpha$ *and* β *are resolvable*$\}$

Intuitively, $upd(\varphi, \ell)$ encodes the CNF-state after execution of an action, that causes ℓ to be true, in φ.

For a PI-state φ, one can prove that every resolvent in $\varphi_\ell|\varphi_{\bar{\ell}}$ either is trivially redundant for $(\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}))$ or already exists in this set. Thus, the update of PI-state φ is as

$$upd_{pi}(\varphi, \ell) = (\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}})) \wedge \ell$$

One can also prove that upd_{pi} of a PI-state results in a PI-state. Furthermore, the result of updating a PI-state (resp. CNF-state) φ w.r.t. a consistent set of literals L is independent from the order in which the literals in L are introduced. For a consistent set of literals L, we define $upd_{pi}(\varphi, L) = upd_{pi}(upd_{pi}(\varphi, \ell), L \setminus \{\ell\})$ for any $\ell \in L$ if $L \neq \emptyset$ and $upd_{pi}(\varphi, \emptyset) = \varphi$. For a CNF-state φ, similarly, we denote $upd(\varphi, \emptyset) = \varphi$ and $upd(\varphi, L) = upd(upd(\varphi, \ell), L \setminus \{\ell\})$ for any $\ell \in L$ if $L \neq \emptyset$.

Definition 1. *Let φ be a PI-state and γ be a consistent set of literals. The* enabling form *of φ w.r.t. γ, denoted by $\varphi \oplus \gamma$, is a PI-belief state defined by*

$$\varphi \oplus \gamma = \begin{cases} \{\varphi\} & \text{if } \varphi \models \gamma \text{ or } \varphi \models \neg\gamma \\ \{PI(\varphi \wedge \gamma), PI(\varphi \wedge \neg\gamma)\} & \text{otherwise} \end{cases}$$

where $\neg\gamma$ is the clause $\{\bar{\ell} \mid \ell \in \gamma\}$.

It is easy to see that $\varphi \oplus \gamma$ is a set of (at most two) PI-states such that for every $\delta \in \varphi \oplus \gamma$, $\delta \models \gamma$ or $\delta \models \neg\gamma$. For a PI-belief state Ψ, let $\Psi + \gamma = \bigcup_{\varphi \in \Psi} (\varphi \oplus \gamma)$. For an outcome o_i of action a and a PI-state φ, let $enb_{pi}(o_i, \varphi) = ((\varphi \oplus \psi_1) \oplus \ldots) \oplus \psi_k$ where $o_i = \{\psi_1 \rightarrow \ell_1, \ldots, \psi_k \rightarrow \ell_k\}$.

Definition 2. *Let φ be a PI-state and a be an action. The progression function between PI-states, denoted by $\Phi_{PI}(a, \varphi)$, is defined as follows:*

- $\Phi_{PI}(a, \varphi) = \otimes[\bigcup_{o_i \in O(a)} \{upd_{pi}(\chi, e(o_i, \chi)) \mid \chi \in enb_{pi}(o_i, \varphi)\}]$ *if $\varphi \models pre(a)$; and*
- $\Phi_{PI}(a, \varphi) = \bot$ *otherwise.*

Given a fluent formula φ, by $BS(\varphi)$ we denote the belief state represented by φ. By definition, for a sensing action ω, the execution of ω in $BS(\varphi)$ results in two disjoint belief states S_1 and S_2 such that $S_1 \cup S_2 = BS(\varphi)$, $S_1 \models \ell(\omega)$, and $S_2 \models \overline{\ell(\omega)}$. Observe that $S_1 \equiv \varphi \wedge \ell(\omega)$ and $S_2 \equiv \varphi \wedge \overline{\ell(\omega)}$. Thus, the execution of ω in a PI-state φ results in two PI-states: $\varphi^+_\omega = PI(\varphi \wedge \ell(\omega))$ and $\varphi^-_\omega = PI(\varphi \wedge \overline{\ell(\omega)})$.

Similarly to the definition of $\widehat{\Phi}$, we define $\widehat{\Phi_{PI}}$, an extended progression function that maps a transition tree and a PI-state to a set of PI-states, by replacing each belief state S with its encoding PI-state φ, where, for each observation ω in Ω (last item), φ^+_ω and φ^-_ω play the role of S^+_ω and S^-_ω, respectively. The correctness of $\widehat{\Phi_{PI}}$ is given next.

Theorem 1. *Let φ be a PI-state and T be a transition tree. Then each belief state in $\widehat{\Phi}(T, BS(\varphi))$ is equivalent to a PI-state in $\widehat{\Phi_{PI}}(T, \varphi)$, and each PI-state in $\widehat{\Phi_{PI}}(T, \varphi)$ represents a belief state in $\widehat{\Phi}(T, BS(\varphi))$.*

Thus, $\widehat{\Phi_{PI}}$ is equivalent to the complete semantics of $\widehat{\Phi}$.

Similar to defining Φ_{PI}, the definition of the progression function for CNF-states, denoted by Φ_{CNF} and used in the development of $\text{C}\textsc{nf}_{ct}$, is presented in (To *et al.* 2011b).

The Planner PI_{ct}

Implementation of PI_{ct}: We built PI_{ct} on top of PIP (To *et al.* 2010b) using the (extended) prime implicates representation. Like $\text{C}\textsc{nf}_{ct}$ and $\text{D}\textsc{nf}_{ct}$, PI_{ct} employs the same AND/OR search algorithm PrAO (To *et al.* 2011a) for contingent planning and the same input language extended from that of PIP to allow non-deterministic and sensing actions.

Heuristics: For a better understanding of the effectiveness of the representations, PI_{ct} uses the same heuristic function as $\text{C}\textsc{nf}_{ct}$ and $\text{D}\textsc{nf}_{ct}$ do, based on the number of satisfied subgoals and the number of known literals in the belief state. Observe that the second component of the heuristic function prioritizes expansion of nodes with less uncertainty and smaller size in all the representations. Note that the use of the same search framework, i.e., same search algorithm and heuristic function, allows the planners to expand/generate the same sets of nodes (belief states) in the search graph.

Empirical Performance: We compare PI_{ct} with $\text{C}\textsc{nf}_{ct}$, $\text{D}\textsc{nf}_{ct}$, CLG, contingent-FF, and POND 2.2 on a large set of benchmarks. These planners are known to be among the best available contingent planners. We executed contingent-FF with both available options (with and without helpful actions) and report the best result for each instance. POND was executed with AO* search algorithm (aostar). For CLG, we observed that the translation time can vary in a very wide range on each instance so we report the average result of several execution times. All the experiments were performed on a Linux Intel Core 2 Dual 9400 2.66GHz workstation with 4GB of memory with the time-out limit of two hours.

Most of the benchmarks have been collected from the contingent-FF distribution ($btcs$, $btnd$, bts, $ebtcs$, $egrid$, $elogistic$, and $unix$) and from the CLG distribution ($cball$, $doors$, $localize$, and $wumpus$). The others including $e1d$, ecc, $edisp$, and $epush$ are variations of the challenging conformant domains 1-*dispose*, *corner-cube*, *dispose*, and $push$, respectively. These domains are modified by us to force planners to generate conditional plans as the new problems do not have a conformant plan.

Tables 1 reports the overall performance of PI_{ct} in comparison with $\text{C}\textsc{nf}_{ct}$/$\text{D}\textsc{nf}_{ct}$ and the other state-of-the-art contingent planners. The columns 2, 3, and 4 report the total run-time in seconds for PI_{ct}, $\text{C}\textsc{nf}_{ct}$, and $\text{D}\textsc{nf}_{ct}$ respectively. Since PI_{ct}, $\text{C}\textsc{nf}_{ct}$, and $\text{D}\textsc{nf}_{ct}$ use the same search framework, they return the same solution for each problem (except for the case that one finds a solution but another produces out-of-memory or time-out for the same problem). Therefore we report in the 5th column the size s (number of actions) and the depth d of the solutions for these planners.

Problem	PIct	CNFct	DNFct	s/d	CLG: t (s/d)	cont-FF: t (s/d)	Pond: t (s)
btcs-70	83.59	111	**2.76**	139/70	331 (140/140)	123 (139/70)	74 (139)
btcs-90	247	347	**5.63**	179/90	503 (180/180)	477 (179/90)	TO
btnd-70	43	38.12	**1.47**	209/72	NA	536 (140/72)	TO
btnd-90	119.5	112.4	**2.28**	369/92	NA	2070 (180/92)	TO
bts-70	79.6	102	**1.53**	139/70	618 (70/70)	1672 (70/70)	TO
bts-90	235	319	**2.6**	179/90	634 (90/90)	TO	TO
cball.3-4	**35**	65.2	346	93.8k/71	761 (1.3M/61)	TO	572 (35k)
cball.5-2	**4.38**	6.97	22.5	5169/107	167 (72.8k/107)	TO	OM
cball.5-3	**96**	206	OM	134k/166	TO	TO	OM
cball.9-1	29.4	32	**23.5**	365/193	113 (3385/197)	TO	OM
cball.9-2	**306**	581	OM	43k/374	TO	TO	OM
doors-7	**5.14**	5.64	5.27	2193/53	7.6 (2153/51)	E	18 (2159)
doors-9	**46.98**	63.3	58.8	45k/89	585 (46k/95)	E	1262(44k)
doors-11	OM	**1429**	OM	1.1M/124	TO	E	TO
e1d-3-5	**80.3**	118	191	296k/183	TO	TO	TO
e1d-5-2	**3.71**	5.44	4.31	2887/132	1382 (13k/99)	E	TO
e1d-5-3	**54.4**	93.8	91.5	77k/407	TO	TO	TO
e1d-9-1	31.3	33.8	**21.9**	324/184	TO	TO	TO
e1d-9-2	320	578	**165**	32k/744	TO	TO	TO
ebtcs-70	36.8	30.6	**1.04**	139/70	24.79 (209/71)	63 (139/70)	24.7 (139)
ebtcs-90	107	97	**1.56**	179/90	69.99 (269/91)	255 (179/90)	TO
ecc40-20	2.04	**1.15**	**1.15**	466/70	2089 (275/75)	37.1 (288/63)	TO
ecc75-37	7.75	**3.04**	3.24	903/202	TO	999 (529/114)	TO
ec119-59	25	**8.26**	9.18	1.4k/290	TO	TO	TO
edisp.3-5	**84.6**	134	194	335k/90	TO	E	TO
edisp.5-2	3.6	5.33	**2.91**	2.7k/97	27.4 (8k/87)	E	TO
edisp.5-3	**50.3**	91	84	74k/139	1588 (266k/112)	E	TO
edisp.9-1	29.6	31.6	**20.4**	325/177	140 (1051/237)	E	TO
edisp.9-2	293	527	**164**	24k/320	TO	E	TO
egrid-3	**1.8**	2.06	**1.8**	352/47	1180 (111/28)	943 (58/41)	105 (148)
egrid-4	**3.01**	3.73	3.06	849/54	1558 (884/48)	TO	OM
egrid-5	**9.67**	13	10.05	1.4k/136	657 (208/40)	TO	OM
elog.-7	0.94	0.98	0.9	416/126	0.11 (210/22)	**0.04** (223/23)	0.95 (212)
elog.-L	OM	TO	OM		**90** (36152/73)	TO	OM
epush3-5	**25.4**	38.9	123	33k/149	TO	TO	TO
epush3-6	1384	**495**	OM	262/179	TO	TO	TO
epush6-2	11	12.1	**9.93**	5k/241	447 (24.5k/148)	TO	TO
epush6-3	**183**	310	362	103k/383	TO	TO	TO
epsh10-1	75.5	79.4	**54.9**	864/345	342 (1983/446)	TO	TO
epsh10-2	659	1151	**399**	65k/839	TO	TO	TO
local.-5	1.66	0.64	**0.54**	48/31	0.57 (112/24)	42 (53/53)	TO
local.-9	30.7	3.6	**0.82**	110/63	12.8 (386/50)	MC	TO
local-13	346	46.9	**2.17**	289/186	OM	MC	TO
unix-2	0.66	0.68	0.65	48/37	0.35 (50/39)	**0.13 (48/37)**	1.71 (48)
unix-3	2.04	2.32	**1.87**	111/84	4.93 (113/86)	3.84 (111/84)	OM
unix-4	18.9	21.3	**17.1**	238/179	78.9 (240/181)	143 (238/179)	OM
wump.-5	2721	2.34	**2**	1.3k/43	5.61 (754/41)	E	4.65 (587)
wump.-7	TO	61.1	**56.4**	38k/86	91.6 (6552/57)	E	TO
total: 48	16/45	5/47	26/43		1/30	2/15	0/9

Table 1: Comparing $\text{PI}_{ct}$with $\text{C}\textsc{nf}_{ct}$/$\text{D}\textsc{nf}_{ct}$ and other planners. TO: *time-out*; OM: *out-of-memory*; NA: *not supported* (CLG does not support non-deterministic actions); E: *incorrect report*; MC: *too many clauses* (for contingent-FF to handle).

In the last three columns of the other planners—written as t (s/d)—t, s, and d denote the total run-time in seconds, the solution size, and the solution depth, respectively (POND does not report the depth information). Usually, d and s are criteria for evaluating the quality of a solution. We consider d to be more important, as it is the maximum number of actions to be executed to obtain the goal. The last row summarizes the performance of the planners in the format n_1/n_2, where n_1 denotes the number of solutions the planner found quickest and n_2 is the number of instances (out of total 48 instances) the planner is able to find a solution for.

We observed that there are several problems for which the experimental results differ from those reported in the literature. We suspect that the versions of the other planners we downloaded perform differently than their predecessors, and/or the environments for conducting the experiments are different (e.g., different hardware/OS).

Table 1 shows that the quality of the solutions found by PI$_{ct}$, CNF$_{ct}$, and DNF$_{ct}$ is, in general, comparable to those found by the other planners. However, each of our planners performs clearly much better than the others on most domains, except $elogistic$. We believe that the heuristic scheme used in our planners is not suitable for this domain. The summary results in the last row show that, within a large set of benchmarks, DNF$_{ct}$ is fastest with the greatest value of n_1, CNF$_{ct}$ is the most scalable planner that can solve most instances (47 out of 48), and PI$_{ct}$ is in between. We will investigate the reason that PI$_{ct}$ is faster but able to solve less instances compared with CNF$_{ct}$ in the next section[1].

Effectiveness: Prime Implicates v.s. CNF

This section analyzes how prime implicates and CNF representations affect the performance and scalability of the corresponding planners, identifies the reasons why their relative effectiveness varies considerably across classes of problems.

To explain the good scalability of PI$_{ct}$ and CNF$_{ct}$, we first consider the cases of *cball-n-m*, *e1d-n-m*, *edis-n-m*, and *epush-n-m* in which PI$_{ct}$, CNF$_{ct}$, and DNF$_{ct}$ perform best. In these domains, n and m denote the number of locations and the number of objects, respectively. The location of each object o_i ($i = 1, \ldots, m$) is initially unknown among n given places p_j ($j = 1, \ldots, n$), and is described by the literal $at(o_i, p_j)$. The number of states in the initial belief state, hence, is linear in n^m, i.e., exponential in m. On the other hand, the size of the initial PI-state/CNF-state in these domains is linear in $m \times n^2$, i.e., linear in m. This explains why PI$_{ct}$ and CNF$_{ct}$ can scale best when m increases if n is not too large and DNF$_{ct}$ perform best when n is large and m is small enough (1 or 2). Observe that the performance of CLG is comparable to our planners when $m = 1$.

In general, the performance and scalability of PI$_{ct}$ and CNF$_{ct}$ are highly competitive due to the compact representation[2] and the efficient progression function implemented in each of them. However, PI$_{ct}$ is faster while CNF$_{ct}$ is more scalable, as observed earlier. When the size of PI-states and the size of CNF-states are comparable, Φ_{PI} is much faster than Φ_{CNF} for the following reasons:

- **Checking satisfaction** of a literal or clause in a PI-state is linear in the number of clauses of the PI-state, while it is exponential in a CNF-state.
- **Update function**: It is easy to see that upd_{pi} is much faster than upd since upd needs to compute the resolvent clauses and the $min(.)$ function but upd_{pi} does not.
- **Cross-product**: The reduced-cross-product function $\otimes$ used to convert a PI-belief state to the equivalent PI-state is faster than the min-cross-product function $\otimes_{min}$ used for CNF-states.
- $PI(.)$ **v.s.** $min(.)$: These functions used to convert a CNF-formula to the equivalent PI-state and CNF-state, respectively. In general, $PI(.)$ is exponential while $min(.)$ is polynomial (in the size of the respective formula). However, a complete computation of the PI-state from a CNF-formula is needed only once for the initial belief state. During the search, the PI-states can be computed incrementally, mostly when adding a literal or clause to a PI-state, using an incremental algorithm for computing prime implicates. This computation is usually very fast, even much faster than $min(.)$ in most problems as observed in the experiments.

One will be able to observe that this analysis agrees with the empirical results presented later.

On the other hand, there are problems where the PI-states are much larger than the equivalent CNF-states and/or $PI(.)$ generates too many intermediate non-prime implicates and takes so much time. In this case, PI$_{ct}$ does not perform well and may cause out-of-memory, while CNF$_{ct}$ does not.

For a better understanding of the performance of PI$_{ct}$ and CNF$_{ct}$ on the benchmarks, we report the size of PI-states/CNF-states and a breakdown of their execution time (in seconds) w.r.t the aforementioned operations of Φ_{PI} and Φ_{CNF} in Table 2. The first column—in the format $n_1/n_2|n_3/n_4$—reports the number of non-unit clauses in the initial PI-state (n_1) and the initial CNF-state (n_2), and the average number of non-unit clauses in a PI-state (n_3) and CNF-state (n_4), respectively. If $n_1 = n_2$ (resp. $n_3 = n_4$) then we omit one. Since the computation cost of Φ_{PI} (resp. Φ_{CNF}) and the memory consumption depends mostly on the non-unit clauses in the PI-state (resp. CNF-state), only non-unit clauses are accounted for in Table 2. The second column denotes the search time of PI$_{ct}$ (left) and CNF$_{ct}$ (right) as the translation time of the input theory for both planners are the same and independent from the representation used. The time PI$_{ct}$ spends to compute the initial PI-state from the initial CNF-formula is reported in the third column. The last four columns report the computation time of PI$_{ct}$ (left) and CNF$_{ct}$ (right) for conversion of PI-states/CNF-states, checking satisfaction, updating formulae, and cross-production process respectively. Note that these items are not disjoint, e.g., the cross-production process for CNF$_{ct}$ ($\otimes_{min}$) also incurs the computation of $min(.)$ function. On the other hand, there are such operations that consumes a significant amount of time as writing to/reading from memory and heuristic evaluation that are not reported here. This explains why the breakdown times do not add up and they are much smaller for PI$_{ct}$ than for CNF$_{ct}$ but the overall search time is not that different, on several instances, e.g., $unix$-4.

[1]A detailed comparison between CNF representation (CNF$_{ct}$) and DNF representation (DNF$_{ct}$) is given in (To *et al.* 2011b).

[2]Note that the compactness of a representation affects not only the scalability, but also the performance (in term of run-time) of the planner as the larger the formula, the more the computation and the more the memory consumption, i.e., the slower the system.

Problem	# of clauses	search	init	inc. PI(.)/	sat. check	update	x-prod
	init | average	PIct/ CNFct	PI(.)	min(.)	PIct/ CNFct	upd_{pi}/ upd	$\otimes$/ $\otimes_{min}$
btcs-90	4006 | 679	246/ 346	5.86	71.2/ 184.6	0.37/ 29.5	0.02/ 38.8	31.9/ 112
btnd-70	2347 | 215	42.3/ 37.4	2.03	14.6/ 11.2	0.1/ 3.95	0.006/ 1.27	0.56/ 2.87
bts-70	2416 | 412	78.8/ 101	2.13	25.4/ 47	0.15/ 9.38	0.008/ 8.94	10.7/ 32.5
cball5-3	654 | 19.7	94.8/ 204	0.09	8.4/ 141.3	29.4/ 46.6	0.55/ 43.77	0.4/ 37.4
doors-9	148 | 5.62	23.3/ 39.6	0.01	0.55/ 6.78	11.4/ 19.4	0.069/ 2.26	0.16/ 4.6
doors-11	280 | 7.76	OM/ 1324	?	?/ 216.9	?/ 649.3	?/ 69.48	?/ 144
e1d-5-3	903 | 13.2	52.96/ 92.4	0.3	3.7/ 48.08	17.1/ 23.6	0.42/ 15.45	0.41/ 17.2
edisp9-2	6482 | 199	270.7/ 505	15.4	14.57/ 477	24.7/ 57.2	0.3/ 128.6	0.19/ 116
egrid-5	10 | 0.04	7.92/ 11.3	0.00	0.016/ 0.7	3.8/ 5.73	0.09/ 0.52	0.19/ 0.86
epush10-2	9902 | 74	589/ 1081	36.82	16.9/ 684	151.5/ 239	0.72/ 180	0.81/ 222
local.-11	2851 | 372/134	105/ 12.1	2.96	47.7/ 2.26	0.38/ 1.13	0.004/ 0.07	30.8/ 6.9
unix-4	1771| 461	3.46/ 5.86	1.14	0.11/ 2.32	0.64/ 2.55	0.001/ 0.6	0.001/ 0.7
wump.-5	16k/88 | 82/6.8	2719/ 0.92	2711	2.74/ 59	0.29/ 0.28	0.01/0.2	0.01/0.21

Table 2: The size of PI-states/CNF-states and the execution time breakdown of PI_{ct}/CNF_{ct} on different benchmarks.

The experiments reveal that, in most domains, the initial PI-state is the same as the initial CNF-state. The only exception is $wumpus$, where the initial PI-state is much larger than the corresponding initial CNF-state, e.g., 15,866 vs. 88 non-unit clauses for $wumpus$-5. On this domain, PI_{ct} spends most of the execution time for the computation of the initial PI-state (2,711 seconds for $wumpus$-5 and time-out for $wumpus$-7). The experiments also show that, in most domains, the set of expanded/generated PI-states is identical to the set of expanded/generated CNF-states, except $wumpus$ and $localize$. In these two domains, the PI-states are much larger than the corresponding CNF-states, making the performance of PI_{ct} much worse than CNF_{ct}. Observe that although the sets of expanded PI-states and CNF-states are the same on *btnd*, CNF_{ct} outperforms PI_{ct} due to the fact that the computation of the PI-states ($PI(.)$) takes too much time. There is an interesting case of $doors$, where PI_{ct} is the best on the smaller instances, but has out-of-memory on $doors$-11 while CNF_{ct} does not, even though both planners expanded the same set of formulae. We suspect that $PI(.)$ generates too many intermediate implicates and/or the PI-states generated by PI_{ct} are much larger than that by CNF_{ct}. $epush$-3-6 is an exception where CNF_{ct} outperforms PI_{ct} even though the breakdown information supports PI_{ct} and PI_{ct} is better on the smaller instances of this domain.

In the other cases, PI_{ct} is observed to outperform CNF_{ct}.

Conclusion

This paper proposed a new approach to contingent planning using prime implicates as a belief state representation and implemented it in the new planner PI_{ct}; and investigated the effectiveness of this representation in comparison with CNF representation, by means of PI_{ct} and CNF_{ct}. Both planners employs the same search framework.

The investigation showed that the belief state representation plays an important role in the context of contingent planning, as both planners provide highly competitive performance in a wide range of benchmarks, even when using an unsophisticated heuristic scheme; and they perform differently through all benchmarks even in the same search framework. The paper identified the advantage and weakness of each representation and the classes of problems that promote or degrade the corresponding planner. In summary, PI_{ct} can find a solution faster, while CNF_{ct} scales better on the size of the problem and requires less memory. When the sizes of the two representations are comparable, the use of prime implicates results in a better performance. However, there are few problems where the prime implicate representation is much larger or PI_{ct} generates so many intermediate implicates, resulting in the poor performance in term of execution time and/or scalability. Fortunately, as shown in the experiments, in most domains, the sets of formulae encoding the belief states in both representations are almost the same.

Finally, we would like to note that the comparison on the planners PI_{ct}, CNF_{ct}, and DNF_{ct} helps evaluate the effectiveness of different belief state representations as they use the same search framework. It also shows that designing a compact yet efficient and scalable belief state representation for contingent planning is an important and difficult task. As such, a systematic comparison—similar to the study in this paper—between these representations with other representations (e.g., DNNF or OBDD) using the same search framework will be very useful. To make this feasible, we will need to define a direct, complete, and efficient progression function—similar to Φ_{PI} and Φ_{CNF}—employing each respective representation. To the best of our knowledge, such a definition has not been proposed. We plan to tackle this issue in our future work. A detailed analysis of memory consumption and time requirement will also be needed.

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A Translation-based Approach to Contingent Planning. *IJCAI.*

P. Bertoli et al. 2001. MBP: a model based planner. *IJCAI Workshop on Planning under Uncert. and Incomplete Inf.*.

Hoffmann, J. and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS*.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318.

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

Haslum, P. and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *Proc. of ECP-99, Lect. Notes in AI Vol 1809.* Springer.

Peot, M. and Smith, D. 1992. Conditional nonlinear planning. *AIPS*.

To, S. T.; Pontelli, E.; and Son, T. C. 2009. A Conformant Planner with Explicit Disjunctive Representation of Belief States. In *ICAPS*.

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

To, S. T.; Son, T. C.; and Pontelli, E. 2010b. On the Use of Prime Implicates in Conformant Planning. In *AAAI*.

To, S. T.; Son, T. C.; and Pontelli, E. 2011a. Contingent Planning as AND/OR forward Search with Disjunctive Representation. In *ICAPS*.

To, S. T.; Pontelli, E.; Son, T. C. 2011b. On the Effectiveness of CNF and DNF Representations in Contingent Planning. In *IJCAI*.