On the Effectiveness of CNF and DNF Representations in Contingent Planning

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

Abstract

This paper investigates the effectiveness of two state representations, CNF and DNF, in contingent planning. To this end, we developed a new contingent planner, called CNF_{ct}, using the AND/OR forward search algorithm PrAO [To *et al.*, 2011] and an extension of the CNF representation of [To *et al.*, 2010] for conformant planning to handle non-deterministic and sensing actions for contingent planning. The study uses CNF_{ct} and DNF_{ct} [To *et al.*, 2011] and proposes a new heuristic function for both planners. The experiments demonstrate that both CNF_{ct} and DNF_{ct} offer very competitive performance in a large range of benchmarks but neither of the two representations is a clear winner over the other. The paper identifies properties of the representation schemes that can affect their performance on different problems.

1 Introduction

Contingent planning [Peot and Smith, 1992] is the task of generating conditional plans in the presence of incomplete information, uncertain action effects, and sensing actions. A contingent plan guides the agent to act conditionally to achieve the goal from every possible initial state of the world and independent from the actual action effects. Contingent planning has been known as one of the hardest problems in planning [Baral *et al.*, 2000; Rintanen 2004].

One of the most common and successful approaches to contingent planning is to transform it into an AND/OR search problem in the belief state space. Significant progress in this direction has been made as state-of-the-art contingent planners (e.g., MBP [Bertoli et al., 2006], POND [Bryce *et al.*, 2006], and contingent-FF [Hoffmann and Brafman, 2005]) can solve problems of large size at different levels of hardness. However, these planners cannot scale up well, mostly due to the underlying belief state representation they employed. For example, the representation using *binary decision diagrams* (BDDs) [Bryant, 1992], used in MBP and POND, is usually very large and sensitive to the order of the variables; furthermore, computing the successor belief state in BDD representation is expensive as it requires intermediate formulae of exponential size. On the other hand, the implicit representation of belief states used in contingent-FF through the action sequences that lead to them from the initial belief state incurs an excessive amount of repeated computation. Moreover, checking whether a proposition holds after the execution of even one single action is co-NP complete.

Later, [Albore, Palacios, and Geffner, 2009] introduced a new approach, that transforms a contingent problem into a search problem in the state space, whose literals represent the beliefs over the original problem, assuming that the uncertainty lies only in the initial world. This method is more efficient as their planner CLG is capable of solving harder problems of larger size. However, the number of literals in the translated problem can be exponential in the number of unknown literals in the original problem, making the state space extremely large and preventing the planner to scale up. Furthermore, CLG does not consider non-deterministic actions.

Recently, in [To *et al.*, 2011], we introduced a new approach to contingent planning which relies on DNF, the belief state representation developed in [To *et al.*, 2009] for conformant planning. We extended 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 on most benchmarks in term of both execution time and scalability on the size of problems. However, we observed that the performance of DNF_{ct} is not as good on the problems where the size of disjunctive formulae encoding the belief states is too large. In some cases, the planner even hardly starts the search due to large size of the formula encoding the initial belief state. This raises the question of whether a CNF representation of belief states would yield a planner with better scalability.

The goal of this paper is first to address the above question. We begin with the development of a new contingent planner, called CNF_{ct}, which employs the CNF representation of belief states for conformant planning [To *et al.*, 2010]. This requires an extension of the transition function developed in that paper to handle non-deterministic and sensing actions for contingent planning. To facilitate the comparisons on belief state representation, we implement CNF_{ct} using the same AND/OR forward search algorithm PrAO and propose a new heuristic scheme to use in this paper for both CNF_{ct} and DNF_{ct}. We also evaluate CNF_{ct} against other state-of-the-art contingent planners besides DNF_{ct}. The experiments show that CNF_{ct} performs reasonably well comparing to DNF_{ct} and is better than other planners in many problems. Afterwards, we investigate the effectiveness of the two representations of belief states in contingent planning by identifying their properties and criteria that affect the performance over different classes of problems. The results of this investigation support our empirical evaluation which shows that neither of the two representations dominates the other across all problems.

2 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 (sensing actions), I describes the initial state, and G describes the goal. A and Ω are separate, i.e., $A \cap \Omega = \emptyset$. A *literal* is either a proposition $p \in F$ or its negation $\neg p$. $\overline{\ell}$ denotes the complement of a literal ℓ—i.e., $\overline{\ell} = \neg \ell$, where $\neg\neg p = p$ for $p \in F$. For a set of literals L, $\overline{L} = \{\overline{\ell} \mid \ell \in L\}$. We will often use a set of literals 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) letter 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 effect of the form $\psi \rightarrow \ell$ (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 mutual exclusive, i.e., the execution of a makes one and only one outcome in $O(a)$ occurs. 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 transition 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 $S \models pre(a)$; $\Phi(a, S) = undefined$, otherwise.

Example 1. *Consider a domain $F = \{alive\}$, a singleton belief state $S = \{\{alive\}\}$, an action shoot with $pre(shoot) = \{true\}$ and $O(shoot) = \{o_1, o_2\}$, where $o_1 = \{\emptyset \rightarrow alive\}$ and $o_2 = \{\emptyset \rightarrow \neg alive\}$. One can easily compute: $res(o_1, \{alive\}) = \{alive\}$, and $res(o_2, \{alive\}) = \{\neg alive\}$. Hence, $\Phi(shoot, S) = \{\{alive\}, \{\neg alive\}\}$.*

Observe that, in Example 1, the non-deterministic action *shoot* 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 $S^-_\omega = \{s \mid s \in S, s \models \overline{\ell(\omega)}\}$.

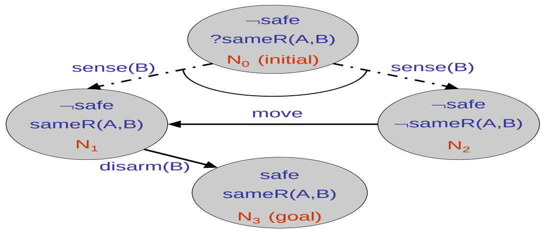

Figure 1: A contingent solution in AND/OR forward search

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. A precise definition is given in the next section. Let us denote $undefined$ by $\bot$. 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 $S = \bot$ or $S = \emptyset \wedge T \neq []$ then $\widehat{\Phi}(T, S) = \bot$; 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 $\widehat{\Phi}(T, S) = \bot$ 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 of the resulting empty belief state is empty.

Let S_I be the set of all possible states satisfying the initial description I. A transition tree T is said to be a solution of P if every belief state in $\widehat{\Phi}(T, S_I)$ satisfies the goal G.

Example 2. *We have an agent A and a bomb B, whose location is unknown in a house with two rooms. A can move around and disarm B if they are in the same room. A can sense whether B is in the same room. The problem P is given as $F = \{sameR(A, B), safe\}$, $A = \{move, disarm(B)\}$, $\Omega = \{sense(B)\}$, $I = \neg safe$, and $G = safe$; where $move = (true, \{sameR(A, B) \rightarrow \neg sameR(A, B), \neg sameR(A, B) \rightarrow sameR(A, B)\})$, $disarm(B) = (sameR(A, B), \{true \rightarrow safe\})$, and $\ell(sense(B)) = sameR(A, B)$. A solution tree for the problem is "$sense(B)(disarm(B) \mid move \circ disarm(B))$". (Fig. 1).*

3 CNF/DNF Representations of Belief States

We will first review several notions of the CNF representation for conformant planning in [To *et al.*, 2010]. We then extend it to deal with non-deterministic actions and observations for contingent planning, in addition to incomplete information.

A *clause* is a set of fluent literals. A clause is *trivial* if it contains the set $\{f, \neg f\}$ for some fluent f. 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 a clause $\alpha \in \varphi$ such that $\ell \in \alpha$. A *unit clause* is a singleton set, i.e. it contains only one literal which is called *unit literal*. A clause α *subsumes* a clause β if $\alpha \subset \beta$. A clause is subsumed by a CNF formula if it is subsumed by some clause in the formula. A CNF formula can be simplified by removing the subsumed clauses from it.

A CNF-formula φ is called a *CNF-state* if

- φ does not contain a trivial clause;
- φ does not contain two clauses γ and δ such that γ subsumes δ; and
- for every unit clause $\{\ell\}$ in φ, $\overline{\ell} \notin \varphi$.

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

In this paper, by $r(.)$, we denote a function that maps a CNF-formula to an equivalent CNF-state. A literal ℓ (resp. a set of literals γ) is true in a CNF-state φ if $\varphi \models \ell$ (resp. $\varphi \models \ell$ for every $\ell \in \gamma$), where $\models$ denotes the standard entailment relation. Also, by φ_ℓ (resp. $\varphi_{\overline{\ell}}$) we denote the set of clauses in φ which contain ℓ (resp. $\overline{\ell}$). Let φ be a CNF formula and ℓ be a literal. By $\varphi - \ell$ we denote the CNF formula obtained by removing every occurrence of ℓ from φ.

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 $\{\alpha_i \cup \beta_j \mid \alpha_i \in \varphi, \beta_j \in \psi\}$. The reduced-cross-product of φ and ψ, denoted by $\varphi \otimes \psi$, is the CNF-state defined by $\varphi \otimes \psi = r(\varphi \times \psi)$. If either φ or ψ is empty then $\varphi \times \psi = \varphi \otimes \psi = \emptyset$.

Note that $\varphi \otimes \psi = \psi \otimes \varphi \equiv \varphi \vee \psi$. For a set of CNF formulae $\Psi = \{\varphi_1, \ldots, \varphi_n\}$, $\otimes[\Psi]$ denotes $\varphi_1 \otimes \varphi_2 \otimes \ldots \otimes \varphi_n$, a CNF-state equivalent to $\bigvee_{i=1}^n \varphi_i$.

Given a CNF-state φ, we define the function *Update* which encodes the CNF-state after the execution of an action, that causes a literal ℓ to be true, in φ as follows.

Let φ be a CNF-state and ℓ a literal. The update of φ by ℓ, denoted by $Update(\varphi, \ell)$, is defined by:

- If ℓ is a unit clause in φ then $Update(\varphi, \ell) = \varphi$.
- If $\overline{\ell}$ is a unit clause in φ, then $Update(\varphi, \ell) = (\varphi - \overline{\ell}) \wedge \ell$.
- Otherwise
  $Update(\varphi, \ell) = r((\varphi \setminus (\varphi_\ell \cup \varphi_{\overline{\ell}})) \wedge \ell \wedge (\varphi_\ell - \ell) \times (\varphi_{\overline{\ell}} - \overline{\ell}))$

Given a consistent set of literals L, we denote $Update(\varphi, \emptyset) = \varphi$ and $Update(\varphi, L) = Update(Update(\varphi, \ell), L \setminus \{\ell\})$ for any $\ell \in L$ if $L \neq \emptyset$.

Let φ be a CNF-state and γ a consistent set of literals. The *enabling form of φ w.r.t. γ*, denoted by $\varphi + \gamma$, is a set of CNF formulae and is defined as

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

where $\neg\gamma$ is the clause $\overline{\gamma} = \{\overline{l} \mid l \in \gamma\}$, $\varphi \wedge \neg\gamma = \varphi \cup \{\overline{\gamma}\}$, and $\varphi \wedge \gamma = \varphi \cup \{\{l\} \mid l \in \gamma\}$.

For a CNF-belief state Ψ, let $\Psi + \gamma = \bigcup_{\varphi \in \Psi} (\varphi + \gamma)$.

The extension starts from this point as follows:

Definition 1. *Let o_i be an outcome of action a. A CNF formula φ is called* 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 CNF formula in Ψ is enabling for o_i.*

For an outcome o_i of action a and a CNF-state φ, let $enb_{o_i}(\varphi) = ((\varphi + \psi_1) + \ldots) + \psi_k$ where $o_i = \{\psi_1 \rightarrow \ell_1, \ldots, \psi_k \rightarrow \ell_k\}$. The effect of a in φ if the outcome o_i occurs, denoted by $e(o_i, \varphi)$, is defined as follows:

$$e(o_i, \varphi) = \{\ell \mid \psi \rightarrow \ell \in o_i, \varphi \models \psi\}.$$

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

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

Given a CNF-state 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 $S_2 \models \overline{\ell(\omega)}$. It is easy to see that $S_1 \equiv r(\varphi \wedge \ell(\omega))$ and $S_2 \equiv r(\varphi \wedge \overline{\ell(\omega)})$. Thus, the execution of ω in φ results in two CNF-states: $\varphi_\omega^+ = r(\varphi \wedge \ell(\omega))$ and $\varphi_\omega^- = r(\varphi \wedge \overline{\ell(\omega)})$.

Similarly to the definition of $\widehat{\Phi}$, we can extend Φ_{CNF} to define $\widehat{\Phi_{CNF}}$, an extended transition function that maps a transition tree and a CNF-state to a set of CNF-states, by replacing each belief state S with its encoding CNF-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.

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

The above theorem shows that $\widehat{\Phi_{CNF}}$ is equivalent to the complete semantics defined by $\widehat{\Phi}$. Thus, any planner using $\widehat{\Phi_{CNF}}$ in its search for solutions will be sound and complete, provided that the search algorithm is sound and complete.

Instead of using CNF-states to represent belief states, [To *et al.*, 2011] uses DNF-states in the development of DNF$_{ct}$. A *partial state* is a consistent set of literals. A *DNF-state* is a set of partial states that does not contain a pair of δ_1 and δ_2 such that $\delta_1 \subset \delta_2$. In developing DNF$_{ct}$, we define the function Φ_{DNF} which is similar to Φ_{CNF}. The definitions leading to the definition of Φ_{DNF} are similar to Definitions 1-2 and are omitted to save space (details are in [To *et al.*, 2011]).

4 Implementation of CNF$_{ct}$

This section describes the implementation of the CNF$_{ct}$. For our comparison, we include a description of the implementation of DNF$_{ct}$. We focus on aspects that help understand better the effectiveness of the CNF and DNF representations employed in CNF$_{ct}$ and DNF$_{ct}$ respectively. Moreover, DNF$_{ct}$ is modified to use the heuristic function implemented in CNF$_{ct}$.

Search Algorithm: As mentioned earlier, CNF$_{ct}$ employs the same AND/OR forward search algorithm PrAO [To *et al.*, 2011] as DNF$_{ct}$ does. We built CNF$_{ct}$ on top of CNF [To *et al.*, 2010]. We modified the implementation of the representation in CNF for non-deterministic and sensing actions.

Heuristics: Both planners employs the same new heuristic function which is based on the number of satisfied subgoals and the number of known literals in the belief state. Let us note that the second component of the heuristic function helps select nodes (CNF-states/DNF-states) of less uncertainty and smaller size in both representations for expansion.

Observe that the use of the same search algorithm and the same heuristic function allows both planners to expand/generate the same sets of nodes (belief states) in the search graph, making the comparison and evaluation of the two representations more accurate.

Data Structure: In DNF$_{ct}$, a partial state is encoded as a *dynamic bitset* available in the standard library of C++. The size of the bitset is the number of literals in the domain. This encoding usually requires less memory for storing partial states and DNF-states, since each literal in a partial state is represented by one bit. Furthermore, the library contains several efficient bitwise operators necessary for the computation of Φ_{DNF}. DNF$_{ct}$ also reduces memory consumption by creating each distinct partial state, needed during the search for solutions, only once, i.e., all identical partial states in different DNF-states are represented by the same dynamic bitset stored in the memory.

In CNF$_{ct}$, since most non-unit clauses contain only two literals as observed in the experiments, each non-unit clause is encoded by a set of integers each of which represents a literal. However, the set of unit literals in each CNF-state is still represented as a bitset. As for DNF$_{ct}$, only distinct clauses (resp. set of unit literals) are created and stored in the memory.

We believe that this data structure scheme is optimal for each representation, making the comparison of the two methods meaningful.

5 Experimental Evaluation

Planners: We compare CNF$_{ct}$ with DNF$_{ct}$, CLG, contingent-FF, and POND 2.2 on a broad set of benchmarks. These planners are known to be among the best available contingent planners. We also considered MBP planner but it does not perform better than the other planners on most benchmarks. Moreover, due to the limit of the column width of the paper, we do not use MBP in this paper. 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 option). The experiments show that the translation time of CLG can vary in a very wide range (from few to hundreds of seconds) for a same instance. Hence, for CLG, we report the best result from several executions of the planner for each instance.

Benchmarks: Among the benchmarks, $btcs$, $btnd$, $ebtcs$, $ebtnd$, $ebts$, $elogistic$, $grid$, $logistics$, and $unix$ come from the contingent-FF distribution, while $cball$, $doors$, $localize$, and $wumpus$ are from the CLG distribution. The other domains, including $e1d$, ecc, $edisp$, and $epush$, are variants 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 they do not have a conformant plan.

All the experiments have been performed on a Linux Intel Core 2 Dual 9400 2.66GHz workstation with 4GB of memory. The time-out limit was set to two hours.

Problem	CNFct/ DNFct	CLG	cont-FF	Pond
btcs-70	113.2/ **2.76** (139/70)	13.7 (140/140)	123.6 (139/70)	74.04 (139)
btcs-90	347/ **5.63** (179/90)	40.7 (180/180)	476.8 (179/90)	TO
btnd-70	40.3/ **1.47** (209/72)	NA	536.6 (140/72)	TO
btnd-90	119.8/ **2.28** (369/92)	NA	2070 (180/92)	TO
cball-3-2	**0.84/** 0.86 (607/34)	2.91 (2641/34)	TO	2.2 (597)
cball-3-4	**65.2/** 346 (93.8k/71)	761 (1.3M/61)	TO	572 (34.8k)
cball-9-1	**32/** 23.5 (365/193)	112.7 (3385/197)	TO	OM
cball-9-2	**580/** OM (43.4k/374)	TO	TO	OM
doors-7	5.64/ **5.27** (2193/53)	7.6 (2153/51)	E	17.99 (2159)
doors-9	63.3/ **58.6** (45k/89)	585 (46k/95)	E	1262 (44k)
doors-11	**1429/** OM (1.1M/124)	TO	E	TO
e1d-3-1	0.55/ **0.54** (33/20)	2.71 (50/20)	E	TO
e1d-3-5	**118/** 191 (296k/183)	TO	TO	TO
e1d-9-1	34/ **22** (324/184)	TO	TO	TO
e1d-9-2	594/ **166** (32.4k/744)	TO	TO	TO
ebtcs-70	30.47/ **1.04** (139/70)	24.79 (209/71)	63 (139/70)	24.69 (139)
ebtcs-90	100/ **1.56** (179/90)	69.99 (269/91)	255.5 (179/90)	TO
ebtnd-70	32.11/ **1.28** (209/72)	NA	16.3 (208/72)	TO
ebtnd-90	99.7/ **1.91** (269/92)	NA	53.15 (268/92)	TO
ebts-70	30.3/ **1.47** (139/70)	16.5 (139/70)	57.9 (139/70)	TO
ebts-90	94.4/ **3.01** (179/90)	44.2 (179/90)	220 (179/90)	TO
ecc-40-20	**1.15/ 1.15** (466/70)	2089 (275/75)	37.1 (288/63)	TO
ecc-75-37	**3.04/** 3.23 (903/202)	TO	999 (529/114)	TO
ecc-99-49	**5.58/** 6.11 (1.1k/184)	TO	5307 (697/150)	TO
ecc-119-59	**8.25/** 9.18 (1.4k/290)	TO	TO	TO
edisp-3-3	1.88/ **1.61** (3.9k/57)	5.77 (8552/52)	E	TO
edisp-3-5	**134/** 194 (335.6k/90)	TO	E	TO
edisp-10-1	72.3/ **47.2** (400/231)	140 (1051/237)	E	TO
edisp-10-2	1479/ **307** (31.8k/404)	TO	E	TO
elogistics-7	0.98/ 0.9 (416/126)	0.11 (210/22)	**0.04** (223/23)	0.95 (212)
elogistics-L	TO/ OM	**90 (36152/73)**	TO	OM
epush-3-1	0.53/ 0.52 (39/31)	**0.39** (50/24)	0.6 (61/37)	TO
epush-3-5	**38.9/** 123 (33.3k/149)	TO	TO	TO
epush-3-6	**504/** OM (262.4k/179)	TO	TO	TO
epush-10-1	80/ **54.7** (864/345)	342 (1983/446)	TO	TO
epush-10-2	1168/ **399** (65k/834)	TO	TO	TO
grid-3	2.24/ 1.74 (383/67)	0.94 (114/30)	**0.06 (23/23)**	104 (178)
grid-4	3.57/ 2.88 (865/69)	4.64 (872/51)	**0.14 (49/49)**	OM
localize-5	0.64/ **0.54** (48/31)	1.86 (112/24)	42 (53/53)	TO
localize-7	1.21/ **0.71** (80/48)	6.89 (231/37)	MC	TO
localize-13	47/ **2.1** (289/186)	OM	MC	TO
logistics-7	1.21/ 1.16 (388/123)	0.25 (31/31)	**0.17** (35/35)	0.97 (178)
logistics-L	TO/ OM	14.34 (96/96)	**7.75 (77/77)**	OM
unix-2	0.68/ 0.65 (48/37)	0.64 (50/39)	**0.13 (48/37)**	1.71 (48)
unix-3	2.34/ **1.87** (111/84)	5.88 (113/86)	3.84 (111/84)	OM
unix-4	22.36/ **17** (238/179)	84.4 (240/181)	143 (238/179)	OM
wumpus-5	2.34/ 1.99 (1.3k/43)	**1.13** (754/41)	E	4.65 (587)
wumpus-7	61.1/ 56.4 (38k/86)	**9.57 (6552/57)**	E	TO

Table 1: Overall performance of CNF$_{ct}$/DNF$_{ct}$ compared with 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).

Overall Evaluation: The performance of CNF$_{ct}$ in comparison with the other state-of-the-art contingent planners is summarized in Table 1. Because of their implementation, CNF$_{ct}$ and DNF$_{ct}$ return the same solution. We therefore report their performance in a single column (2nd column). This column—in the format $t_1/t_2(s/d)$—reports the total runtime in seconds for CNF$_{ct}$ (t_1), DNF$_{ct}$ (t_2), the number of actions in the solution (s), and the depth of the solution tree (d). The other columns—-written as $t(s/d)$—report the total run-time in seconds (t), the number of actions in the solution (s), and the depth of the solution tree (d) for the other planners, whenever this information is available. Observe that s and d are the same for CNF$_{ct}$ and DNF$_{ct}$, as they use the same search algorithm and the same heuristic function. POND does not report the depth information. Usually, d and s are criteria to evaluate the quality of a solution. However, we consider d to be more important, as it is the maximum number of actions the agent needs to execute to achieve the goal.

We observe that there are several problems for which the experimental results differ from those reported in the literature. We suspect that the discrepancy occurs due to the different versions of the other planners and/or the environments for conducting the experiments are different (e.g., different hardware/OS).

As observed from Table 1, CNF_{ct} and DNF_{ct} perform better than other planners

From the experimental results in Table 1 one can observe that CNF_{ct}, like DNF_{ct}, outperforms the other planners while CLG is better than the rest planners in the tested domains. The quality of the solutions found by CNF_{ct}, which is the same as that of DNF_{ct}, is in general not as good as contingent-FF but is better than CLG or Pond in some cases. We believe that the heuristic scheme used in CNF_{ct} plays an important role in this issue. The simplicity of the heuristic function could also be the reason that CNF_{ct} and DNF_{ct} are not as competitive in the domains $elogistics$, $logistics$, and $wumpus$.

6 Effectiveness of CNF/DNF Representations

In this section, we analyze the effectiveness of the CNF and DNF representations on the performance of the respective planners. In particular, we are interested in what features affect the scalability and performance of a planner. From Table 1 one can observe that CNF_{ct} scales best, as it can solve most problem instances on the table, while DNF_{ct} is the fastest planner that solves most instances with shortest time.

Before we get into the details, let us discuss the strength and weakness of each method using the following domains.

Consider *cball*-n-m, $e1d$-n-m, *edis*-n-m, and *epush*-n-m, where n denotes the number of locations and m is the number of objects. Initially, the location of each object o_i ($i = 1, \ldots, m$) is unknown among n given locations p_j ($j = 1, \ldots, n$), and described by the literal $at(o_i, p_j)$. Thus, the number of states in the initial belief state is linear in n^m, i.e., exponential in m. On the other hand, the size of the initial CNF-state is linear in $m \times n^2$. This, we believe, explains why CNF_{ct} scales best when m increases, provided that n is not too large. On the other hand, DNF_{ct} can outperform CNF_{ct} if $n \geq 9$ and $m \leq 2$, e.g., $e1d$-9-1, $e1d$-9-2, *epush*-10-1, etc... The reason is that the CNF-states in these cases is not much smaller or even larger in comparison with the DNF-states, whereas Φ_{DNF} can be computed much faster than Φ_{CNF} due to the difference in their complexity, as analyzed later. Furthermore, the nature of the DNF representation allows to use the bitset encoding compactly along with bitwise operations for computing Φ_{DNF} efficiently.

The above discussion shows that two important issues that can affect a planner's performance and scalability are: (*i*) the size of the formula representing the initial belief state and (*ii*) the efficiency of computing the transition function. For a better understanding of the second factor, let us analyze the complexity of the two transition functions in the worst case.

Let S be a belief state and a be an action. For simplicity, assume that a contains only one outcome o_i.

Complexity of Φ_{CNF}: Let φ be the CNF-state representing S, n be the number of clauses in φ, and m be the length of the largest clause in φ. For simplicity, we assume that n and m are the same for the intermediate CNF-states in the computation of Φ_{CNF}. Let k be the number of non-unit literals in the antecedents of the conditional effects of a and u be the number of the antecedents of the conditional effects of a which are unknown in φ. The computation of Φ_{CNF} incurs the following costs:

1. **Satisfaction checking**: $k \times f(n, m)$, where $f(n, m)$ is the cost of checking satisfaction of a literal in φ. Note that $f(n, m)$ is exponential in n.
2. **Enabling conversion** enb_{o_i}: $2^u \times f_1(n, m)$. Let ψ be one of those antecedents; the conversion of φ to enabling form splits it into $\varphi \wedge \psi$ and $\varphi \wedge \neg\psi$. $f_1(n, m)$ is the cost of conversion of $\varphi \wedge \psi$ or $\varphi \wedge \neg\psi$ to a CNF-state (r(.)), which is polynomial in n and m. There are u of such antecedents, so there will be at most 2^u CNF-states at the end of this process.
3. **Update**: $2^u \times f_2(n, m, e)$, where $e = max\{|e(o_i, \phi)| \mid \phi \in enb_{o_i}(\varphi)\}$ and f_2 is polynomial in n, m, and e.
4. **Reduced-cross-product**: $2^u \times f_3(n, m)$, where f_3 is polynomial in n and m.

The complexity of Φ_{CNF} is $k \times f(n, m) + 2^u \times (f_1(n, m) + f_2(n, m, e) + f_3(n, m))$ which is exponential in n and u.

Complexity of Φ_{DNF}: Let Δ be the DNF-state representing S. The main difference in the complexity of Φ_{DNF} is that checking satisfaction of a literal in Δ is linear in $|\Delta|$ and there is no reduced-cross-product phase, which is often expensive. Moreover, conversion of a DNF-formula to the DNF-state is much simpler than that of a CNF-formula to the CNF-state. In summary, Φ_{DNF} is exponential only in u with much smaller hidden cost compared with Φ_{CNF}.

Empirical Comparison and Discussion: Table 2 presents several features of CNF_{ct} and DNF_{ct} obtained from our experiments. In all columns, cls and $p\text{-}states$ stand for clauses (in CNF_{ct}) and partial states (in DNF_{ct}) respectively. The table contains, beside the size of the formulae representing the initial belief state (3rd column); the average number of clauses, partial states, and unknown antecedents (u) in an expanded CNF-state or DNF-state respectively (4th column); the average size of all clauses/partial states appearing in the expanded CNF-states/DNF-states (5th column); and the number of distinct clauses, partial states and nodes generated by the planners (6th column). We believe that these numbers help explain the scalability and performance of both planners in different domains. Note that, since all the unit literals in a CNF-state are encoded in a bitset and the experiments show that the computation cost of Φ_{CNF} depends mostly on the non-unit clauses in the CNF-state, only non-unit clauses are accounted for in Table 2.

As analyzed earlier, the cost of Φ_{CNF} depends on the number of unknown antecedents u and (non-unit) clauses in the CNF-state and the size of the clauses. The last column reveals that most non-unit clauses in the expanded CNF-states have size two. Hence, the cost of Φ_{CNF} depends mainly on u and the number of clauses in each CNF-state. Observe that (the average of) u is very small ($< 10^{-10}$) in most domains, partly due to the heuristic scheme used in the planners, except

$btcs$, $btnd$, and $localize$ in which DNF$_{ct}$ performs much better than CNF$_{ct}$. The reason is that although both Φ_{CNF} and Φ_{DNF} are exponential in u but Φ_{DNF} does not incur the computation of the reduced-cross-product phase, which is exponential in u, while Φ_{CNF} does; and the enabling conversion of Φ_{DNF} is also much simpler.

The performance of the planners also depends on the memory consumption, which is reflected in the last column. Consider the cases of $ebtcs$ and $ebts$, where DNF$_{ct}$ performs much better than CNF$_{ct}$. The reasons are not only that the average number of clauses of the expanded CNF-states is much bigger than the average number of partial states in an expanded DNF-state, but also the number of distinct clauses generated by CNF$_{ct}$ is much larger than the number of distinct partial states generated by DNF$_{ct}$ for each instance. Now consider the case of $cball$-3-4: even though the average number of partial states of the expanded DNF-states (48.34) is not much larger than the number of clauses (9.71) (remember that Φ_{CNF} is exponential in this number), DNF$_{ct}$ performs much worse than CNF$_{ct}$, because it generates a very large number of distinct partial states (3,643,961), while CNF$_{ct}$ generates only 132 distinct clauses. Observe also that both planners fail to find a solution for $wumpus$-10. While the reason for the failure of CNF$_{ct}$ is only the unsuitable heuristic function, as it generates successor CNF-states quite fast, the failure of DNF$_{ct}$ is mainly due to the large size of the initial DNF-state—DNF$_{ct}$ spends 3230 seconds for the computation of the initial DNF-state from the description of the initial information and produces out of memory during the expansion of the first node. If a better heuristic function for this domain was used, DNF$_{ct}$ would still fail but CNF$_{ct}$ would probably find a solution for $wumpus$-10.

Problem	search time CNF$_{ct}$/DNF$_{ct}$	initial cls/p-states	ave. # of cls/p-states (u)	ave. size of cls/p-states	# of generated cls/p-states (nodes)
btcs-90	346/4.9	4006/90	679/23.4 (.34)	2.03/93	12462/270 (8544)
btnd-90	118.4/1.42	3917/89	351/12.5 (.02)	2.03/183.9	8275/712 (4896)
cc119.59	7.15/8.08	6/8	0.00/1.00 (ϵ)	2.00/357	6/39747 (39787)
cball-3-2	0.286/0.31	36/400	6.33/9.66 (ϵ)	2.27/40	65/5602 (1932)
cball-3-4	64.5/345.6	72/160k	9.71/48.34 (ϵ)	2.27/70	132/3.64M (298k)
cball-9-2	580.8/OM	5868/94.8k	121.33/? (ϵ)	2.07/?	7084/? (202k)
edis-3-2	0.15/0.08	74/81	3.34/2.91 (ϵ)	2.24/32	94/985 (1374)
edis-3-5	133.1/193	185/59k	1.82/5.37 (ϵ)	2.23/65	259/1.05M (1.22M)
edis-10-2	1431/258.7	9.9k/10k	320/29.83 (ϵ)	2.04/305	11.9k/731.8k (166k)
e1d-3-1	0.025/0.013	37/9	5.25/2.61 (ϵ)	2.23/31	44/85 (105)
e1d-3-5	117.6/190.2	185/59k	2.18/6.35 (ϵ)	2.23/61	241/821k (1.05M)
e1d-9-2	570.3/142.7	6482/6561	158.3/17.3 (ϵ)	2.05/493	7.7k/439.5k (166k)
doors-9	39.5/34.94	148/6561	5.62/5.27 (ϵ)	2.2/162	279/249.4k (131.7k)
doors-11	1324/OM	280/161k	7.76/? (ϵ)	2.18/?	516/? (3.1M)
ebtcs-70	29.77/0.39	2416/70	283/12.8 (ϵ)	2.04/70.92	4898/210 (2692)
ebts-90	91.74/0.74	4006/90	679/23.37 (ϵ)	2.03/92	8098/180 (4273)
epush-3-1	0.024/0.011	37/9	4.62/2.41 (ϵ)	2.23/20	44/77 (106)
epush-3-5	38.2/123	185/59k	1.91/12.32 (ϵ)	2.24/60	224/715.4k (377.3k)
epush10.2	1099/330	9.9k/10k	74.26/9.44 (ϵ)	2.04/303	11.4k/740.5k (394k)
local.13	46.54/1.16	5254/103	207/10.3 (5.8)	2.08/109	6527/341 (877)
unix-4	6.05/0.76	1771/60	460/24.38 (ϵ)	2.05/127	1829/2284 (630)
wump.7	53.27/48.52	149/7276	7.39/7.08 (ϵ)	2.17/179.8	330/299k (134140)
wump.10	TO/OM	238/2.56M			

Table 2: The properties of CNF/DNF representations that affect the performance of CNF$_{ct}$/DNF$_{ct}$. $\epsilon < 10^{-10}$, 1k $\approx$ 1000, 1M $\approx 10^6$.

7 Summary

This paper presented a new approach to contingent planning using CNF representation of belief states and implemented it in the planner CNF$_{ct}$. It investigated the effectiveness of CNF and DNF representations by means of CNF$_{ct}$ and DNF$_{ct}$ [To *et al.*, 2011] using a same new heuristic function. The empirical study showed that both methods are very efficient but neither of them dominates the other and their performance can vary significantly on different classes of problems. The paper identified key aspects of each representation that affect the performance of the corresponding planner.

In summary, CNF$_{ct}$ scales better with the size of the problem and requires less memory, while DNF$_{ct}$ can find a solution faster. In addition, we showed that the performance is significantly affected by the total amount of consumed memory, which is determined mainly by the number of distinct clauses/partial states generated, rather than the number of generated CNF-states/DNF-states and the total number of clauses/partial states contained in them.

Acknowledgments

We are grateful to the reviewers for their helpful comments. We also thank Alexandre Albore (one of the authors of CLG planner) for his assistance with the experiments.

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

References

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

[Baral *et al.*, 2000] C. Baral et al. Computational complexity of planning and approximate planning in the presence of incompleteness. *AIJ*, 122:241–267, 2000.

[Bertoli et al., 2006] P. Bertoli et al. Strong planning under partial observability. *Artificial Intelligence*, 170(4-5):337-384, 2006.

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

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

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

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

[Rintanen 2004] J. Rintanen. 2004. Complexity of planning with partial observability. In *ICAPS*, 2004.

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

[To *et al.*, 2010] S. T. To, T. C. Son, and E. Pontelli. A New Approach to Conformant Planning using CNF. In *ICAPS*, 2010.

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