Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)

A New Approach to Conformant Planning using CNF[*]

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

Abstract

In this paper, we develop a heuristic, progression based conformant planner, called CNF, which represents belief states by a special type of CNF formulae, called *CNF-state*. We define a transition function Φ_{CNF} for computing the successor belief state resulting from the execution of an action in a belief state and prove that it is sound and complete with respect to the complete semantics defined in the literature for conformant planning. We evaluate the performance of CNF against other state-of-the-art conformant planners and identify the classes of problems where CNF is comparable with other state-of-the-art planners or scales up better than other planners. We also develop a technique called `one-of` relaxation which helps boost the performance of CNF. We characterize the domains where this technique can be applied and validate this idea by proposing a new set of benchmarks that is really difficult for other planners yet easy for CNF.

Introduction and Motivation

Conformant planning is the planning problem in presence of incomplete information about the initial state. A conformant plan is a sequence of actions that achieves the goal from any possible initial state of the world. For example, given that the execution of an action a changes the value of a proposition f from false to true, regardless of the truth value of other propositions, the sequence containing only the action a is a conformant plan achieving f from any possible state of the world. Conformant planning is important since it removes one of the (unrealistic) assumptions about planning problems: the completeness assumption which states that agent has complete information about the initial state.

Since the problem is introduced (Smith and Weld 1998), several conformant planners have been developed. To date, the most efficient conformant planners are Conformant-FF (CFF) (Brafman and Hoffmann 2004), KACMBP (Cimatti, Roveri, and Bertoli 2004), POND (Bryce, Kambhampati, and Smith 2006), `t0` (Palacios and Geffner 2007), CPA (Tran et al. 2009), and DNF (To, Pontelli, and Son 2009). One of the major difficulties in the building of a conformant planner lies in dealing with the uncertainty about the initial state, which is often represented by a set of formulas describing the possible initial states of the world, which is often referred as the *initial belief state*. In a domain with n propositions, the size of the initial belief state can be 2^n.

The representation of belief states affects the performance of a conformant planner in two ways. First, it can quickly increase the memory usage of the planner, leading to the undesirable "Out of memory" situation, if the size of the belief state is large. Second, it directly influences the time complexity in computing the successor belief states.

Indeed, the literature is rich in methods for representing of and dealing with belief states in conformant planners. CFF does not have a direct encoding of the belief states in its search process. Rather, it stores the sequence of actions reaching to the belief states and recomputes the belief states whenever it is necessary. [1] Checking whether an action is executable needs a call to a SAT-solver. POND uses ordered binary decision diagram (OBDD) (Bryant 1992) in its representation and relies on the OBDD-library to compute the successor belief states. A disadvantage of this approach is that the size of the OBDD can be very large—the structure of the OBDD is sensitive to the ordering of variables and the manipulation of the OBDD might require intermediate OBDDs of exponential size. `t0` translates a conformant planning problem to a classical planning problem, uses a classical planner (FF) to solve the new problem, and then converts the solutions of the new problem to solutions of the original problem. The size of the initial state (the number of the fluents) in the new problem, however, could be exponential to the size of the original problem if completeness is required. CPA identifies and keeps only a set of approximation states, which guarantees completeness of the planner, in its search for solution; thus reducing the size of the belief states, in many cases, significantly. The price CPA has to pay is that the overhead for computing the initial approximation state is sometimes quite significant, leading to its inability to start the search process. DNF goes to an extreme by employing an explicit representation of belief states using disjunctive normal form. DNF expands the DNF representation on demand. As such, it can avoid some problems faced by CPA, CFF, or `t0` and can solve more problems than these planners (To, Pontelli, and Son 2009).

[*]Partial supported by NSF grants IIS-0812267, CBET-0754525, and CREST-0420407.

[1]CFF does implement some techniques which help reduce the number of recomputations.

It is worth mentioning that regardless of the representation of the belief states, *all planners* have difficulties scaling up when the size of the initial belief state is large. For example, for the `coins-21` instance, from the IPC-2006, all planners fail to find a solution either because of "out of memory" error or because of "time out." The initial belief state of this problem contains 10^{16} states, comparing to the "easy instances" in the same domain such as `coins-20`, whose initial belief state contains less than 10^6 possible states, and can be solved by all planners in less than two minutes.

The above discussion motivates us to investigate alternative representation of belief states in conformant planning. More precisely, we develop a conformant planner, called CNF, that uses a specific form of CNF, called *CNF-state*, to represent belief states. We compare CNF with other planners and our experiments show that the new representation does incur some overhead which cannot be easily overcame. In order to exploit the new representation, we develop a new technique called `one-of` relaxation and implement it in CNF. This technique helps further reducing the size of the initial belief state, allowing the planner to solve instances which cannot be solved by other planners due to the size of the initial belief state. We provide a characterization of domains in which this technique can be applied and evaluate this idea by proposing a new set of benchmarks that is hard for the other planners yet easy for CNF.

Background: Conformant Planning

A *planning problem* is 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 $\neg p$. $\overline{\ell}$ 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, $\overline{L} = \{\overline{\ell} \mid \ell \in L\}$. 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. We will often use lowercase (resp. uppercase) letter, possibly with indices, to represent a state (resp. a belief state).

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 domain of P is defined by

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

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 all benchmarks, I is a conjunction of a set of literals, a set of `one-of` statements (resp. `or`-statements)—representing 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`-clauses.

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, for an action a, we will denote with C_a the set of conditional effects of a.

Representing Belief States by CNF Formulae

In this section we develop the theoretical basis for the development of a conformant planner which uses a specific type of CNF formulae, which will be introduced shortly, to represent belief states. Our first task is to define a progression function Φ_{CNF}, similar to the function Φ, for computing the successor belief state.

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. The literal in a unit clause is called *unit literal*. A clause α *subsumes* a clause β (or β is *subsumed by* α) if $\alpha \subset \beta$. A clause is subsumed by a CNF formula if it is subsumed by some clause in the CNF formula. Obviously, if φ' is obtained by removing some subsumed clause(s) in a CNF formula φ then $\varphi' \equiv \varphi$.

Definition 1. *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.

Intuitively, a CNF-state φ is minimal in the sense that it does not contain redundant clauses and can be viewed as a set of states. For example, if $F = \{p, q\}$, the CNF-state $\varphi_1 = \{\{p\}\}$ encodes the set of states in which p is true, i.e., the set $\{\{p, q\}, \{p, \neg q\}\}$; the CNF-state $\varphi_2 = \{\{\neg p\}\}$ encodes the set of states in which p is false; the set of CNF-states $\{\varphi_1, \varphi_2\}$ represents all possible states of the domain. Observe also that even though φ_1 is equivalent to

$\varphi = \{\{p\},\{p,q\}\}$, φ is not a CNF-state as it violates the second condition of Def. 1. Furthermore, observe that a CNF-formula might be represented by several CNF-states. For example, the formula p can be represented by either $\{\{p\}\}$ or $\{\{p,q\},\{p,\neg q\}\}$ [2].

It is easy to see that an arbitrary CNF-formula can be converted into an equivalent CNF-state and there exist different algorithms for this purpose. In CNF, we implemented a fixed-point algorithm which is a modification of an effective CNF-preprocessor using unit-propagation and subsumption techniques. The algorithm is deterministic and polynomial in the size of the formula. Furthermore, it yields a CNF-state that cannot be further simplified using the same techniques. For our discussion, it suffices to note that $r(\varphi)$, the result of the algorithm, is a CNF-state equivalent to φ.

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. $\varphi_{\bar\ell}$) we denote the set of clauses in φ which contain ℓ (resp. $\bar\ell$).

We define some operations for the manipulation of CNF formulae, which will be used in manipulating CNF-states.

Definition 2. *Let φ be a CNF formula and ℓ be a literal. By $\varphi - \ell$ we denote the CNF formula obtained by removing from φ every occurrence of ℓ.*

E.g., $p \wedge (q \vee \neg r) - q = p \wedge \neg r$ and $\neg p \wedge (q \vee r) - \neg p = q \vee r$.

Definition 3. *For two CNF formulae $\varphi = \{\alpha_1,\dots,\alpha_n\}$ and $\psi = \{\beta_1,\dots,\beta_m\}$, 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 $\varphi \times \psi = \emptyset$.*

Observe that $\varphi \times \psi$ is nothing but a transformation of $\varphi \vee \psi$ to a CNF formula. For a set of CNF formulae $\Psi = \{\varphi_1,\dots,\varphi_n\}$, $\times[\Psi]$ denotes $\varphi_1 \times \varphi_2 \times \dots \times \varphi_n$, a CNF-formula 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.

Definition 4. *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 $\bar\ell$ is a unit clause in φ, then $Update(\varphi,\ell) = (\varphi - \bar\ell) \wedge \ell$.*
- *Otherwise* $Update(\varphi,\ell) = r((\varphi \setminus (\varphi_\ell \cup \varphi_{\bar\ell})) \wedge \ell \wedge (\varphi_\ell - \ell) \times (\varphi_{\bar\ell} - \bar\ell))$

Intuitively, $Update(\varphi,\ell)$ encodes Equation (1) when the set of effects is $\{\ell\}$ and the action is executable. As φ represents a belief state $S = \{s \mid s \models \varphi\}$, $Update(\varphi,\ell)$ should be equivalent to the formula $\bigvee_{s \in S}(s \setminus \{\bar\ell\} \cup \{\ell\})$. As such, the first two cases of Definition 4 are obvious. To deal with the third case, first φ is converted to the equivalent formula $\varphi' = (\varphi_0 \wedge (\varphi_\ell - \ell) \wedge (\varphi_{\bar\ell} - \bar\ell)) \vee (\varphi_0 \wedge (\varphi_\ell - \ell) \wedge \bar\ell) \vee (\varphi_0 \wedge (\varphi_{\bar\ell} - \bar\ell) \wedge \ell)$, where $\varphi_0 = \varphi \setminus (\varphi_\ell \cup \varphi_{\bar\ell})$. Then φ' is updated by updating each of its disjuncts: adding ℓ to the first and updating the last two ones according to the first two cases in Definition 4 (note that the first disjunct in φ' does not contain ℓ or $\bar\ell$, the second contains the unit literal $\bar\ell$, and the third contains the unit literal ℓ). The formula in the third case is a simplification of the obtained formula. The definition is illustrated in the next few examples:

- ◦ $Update(\{\{f\},\{g,\neg p\}\},p) = \{\{f\},\{p\}\}$
- ◦ $Update(\{\{f\},\{h,p\}\},p) = \{\{f\},\{p\}\}$
- ◦ $Update(\{\{f\},\{g,\neg p\},\{h,p\}\},p) = \{\{f\},\{p\},\{g,h\}\}$

We can prove the following proposition.[3]

Proposition 1. *Given a CNF-state φ and two literals ℓ_1 and ℓ_2 such that $\ell_1 \neq \bar\ell_2$, then $Update(Update(\varphi,\ell_1),\ell_2) = Update(Update(\varphi,\ell_2),\ell_1)$.*

Proposition 1 shows that the result of updating a CNF-state φ using a consistent set of literals L is independent from the order in which the various literals of L are introduced. 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 $L \neq \emptyset$.

Our goal is now to define the transition function Φ_{CNF}. Given an action a with the precondition $pre(a)$ and its set of conditional effect C_a and a CNF-state φ, we need to define the successor CNF-state $\Phi_{CNF}(a,\varphi)$. Since φ is a CNF-state, when computing $\Phi_{CNF}(a,\varphi)$, for each $\psi \rightarrow \ell$ in C_a, there are three cases that need to be considered: (*i*) $\varphi \models \psi$; (*ii*) $\varphi \models \neg\psi$; and (*iii*) $\varphi \not\models \psi$ and $\varphi \not\models \neg\psi$. As an example, if $\varphi = p \wedge q$ and $\psi = r$, then we have $\varphi \not\models \psi$ and $\varphi \not\models \neg\psi$. As such, to define Φ_{CNF}, we need the following definition.

Definition 5. *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 $\bar\gamma = \{\bar l \mid l \in \gamma\}$, $\varphi \wedge \neg\gamma = \varphi \cup \{\bar\gamma\}$, and $\varphi \wedge \gamma = \varphi \cup \{\{l\} \mid l \in \gamma\}$

It is easy to see that $\varphi + \gamma$ is a set of CNF formulae such that for every $\delta \in \varphi + \gamma$, $\delta \models \gamma$ or $\delta \models \neg\gamma$. For a CNF-belief state Ψ, let $\Psi + \gamma = \bigcup_{\varphi \in \Psi}(\varphi + \gamma)$. It holds that

Proposition 2. *Let φ (resp. Ψ) be a CNF-state (resp. CNF-belief state). If γ is a consistent set of literals, then $\varphi + \gamma$ (resp. $\Psi + \gamma$) is a CNF-belief state equivalent to φ (resp. Ψ). If γ_1 and γ_2 are two consistent sets of literals then $(\varphi + \gamma_1) + \gamma_2 \equiv (\varphi + \gamma_2) + \gamma_1$.*

Definition 6. *Let a be an action with the set of conditional effects C_a. A CNF formula φ is called* enabling *for a if for every conditional effect $\psi \rightarrow \ell$ in C_a, either $\varphi \models \psi$ or $\varphi \models \neg\psi$ holds. A set of CNF formulae Ψ is* enabling *for a if every CNF formula in Ψ is enabling for a.*

For an action a and a CNF-state φ, let $enb_a(\varphi) = ((\varphi + \psi_1) + \dots) + \psi_k$ where $C_a = \{\psi_1 \rightarrow \ell_1,\dots,\psi_k \rightarrow \ell_k\}$.

Proposition 3. *For every CNF-state φ and action a, $enb_a(\varphi)$ is a CNF-belief state equivalent to φ and enabling for a.*

For an action a and a CNF-state φ, the effect of a in φ, denoted $e(a,\varphi)$, is defined as follows. $e(a,\varphi) = \{\ell \mid \psi \rightarrow \ell \in C_a, \varphi \models \psi\}$. We are now ready to define the function Φ_{CNF}.

[2] In the implementation, actually this will be simplified to $\{\{p\}\}$

[3] Due to lack of space, all proofs are omitted. They can be found at `www.cs.nmsu.edu/~sto/full_cnf_paper.pdf`.

Definition 7. *Let φ be a CNF-state and let a be an action. By $\Phi_{CNF}(a, \varphi)$ we denote the transition function for CNF-states:*

- $\Phi_{CNF}(a, \varphi){=}r(\times[\{\mathit{Update}(\phi, e(a, \phi))|\phi{\in}enb_a(\varphi)\}])$ *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 consisting of states satisfying φ. It holds that

Lemma 1. *Let ϕ be a CNF-state enabling an action a. Then,*

$$\mathit{Update}(\phi, e(a, \phi)) \equiv \{s \setminus \overline{e(a, s)} \cup e(a, s) | s \in BS(\phi)\}$$

Let $\widehat{\Phi_{CNF}}$ is the function defined by extending Φ_{CNF} to allow for reasoning about the effects of an action sequence in the same manner of extending Φ to $\widehat{\Phi}$. Using Lemma 1, one can prove the correctness of the following theorem.

Theorem 1. *For every CNF state φ and action sequence α_n, $\widehat{\Phi_{CNF}}(\alpha_n, \varphi) \equiv \widehat{\Phi}(\alpha_n, BS(\varphi))$.*

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.

CNF-States and `one-of` Statements

One of the main methods used in specifying uncertainty in conformant planning benchmarks is the use of the `one-of` construct. This construct provides a way to specify mutual exclusive information about certain proposition (e.g., a truck t is at one (and only one) place p_1, p_2, p_3 is expressed by the statement `one-of`$(at(t, p_1), at(t, p_2), at(t, p_3))$. The size of the initial belief state (the number of states) depends largely on the number of `one-of` statements in the problem specification. The following proposition discusses this issue.

Proposition 4. *A* `one-of` *statement c can be converted to*

- *an equivalent DNF formula of minimum size of $|c|^2$; and*
- *an equivalent CNF formula of minimum size of $\Theta(|c|^2)$.*

where $|c|$ denotes the number of literals in c.

A set $S = \{c_1, \ldots, c_n\}$ of independent `one-of` *statements can be converted to an equivalent DNF formula of minimum size $\Pi_{i=1}^n |c_i|^2$ and to an equivalent CNF formula of minimum size $\Theta(\Sigma_{i=1}^n |c_i|^2)$.*

In the above proposition we use *independent* which will be defined later in the Definition 9. This proposition shows that the representation of belief state affect the size of the initial belief state, which the planner needs to deal with. This is also the main reason behind the development of the `one-of` combination technique by CPA which greatly influences the performance of CPA and DNF. This motivates us to develop an alternative technique to the `one-of` combination technique for use in the CNF planner.

Intuitively, the CNF-representation favors the second way of specifying uncertainty in conformant planning benchmarks, the use of `or` statements. Can we replace `one-of` statements by `or` statements? Surprisingly, in many benchmarks, the answer for this question is yes. We call this technique as `one-of` *relaxation*. To illustrate this technique, let us consider the $dispose(m, n)$ problem, one of the benchmarks in IPC-2008. The goal of this problem is to retrieve m objects $o_1, \ldots, o_m$, whose initial locations are unknown, and placing them in a trash can. The initial information is specified by δ_i = `one-of`$(at(o_i, p_1), \ldots, at(o_i, p_n))$. We say that δ'_i = `or`$(at(o_i, p_1), \ldots, at(o_i, p_n))$ is a relaxation of the `one-of` statement δ_i since the belief state specified by the `or` statement contains all states belonging to the belief state specified by the `one-of` statement. Let the planning problem P' be obtained from the problem $dispose(m, n)$ by replacing δ_i with δ'_i for every i. Obviously, the size of the CNF-state representing the initial state of $dispose(m, n)$ is much larger than the size of the CNF-state encoding the initial state of P' ($m \times \Theta(n^2)$ vs. $m \times n$). Moreover, every solution of P' is also a solution of $dispose(m, n)$. This can be generalized as follows.

Proposition 5. *Let $P = \langle F, O, I, G\rangle$ and $P' = \langle F, O, I', G\rangle$ be two conformant problems such that $I \models I'$. If α is a solution of P', then α is also a solution of P.*

For a CNF-state φ and a set φ' of clauses in φ, we have that $\varphi \models \varphi'$. As such, given that the initial belief state is represented by a CNF-state φ, the above proposition allows one to attempt to solve the problem with a subset of φ. As shown above, this could be significant, especially when there are `one-of` statements that can be replaced by `or` statements.

Definition 8. *Let $P = \langle F, O, I, G\rangle$ be a planning problem. We define $P_{\texttt{one-of}}\langle F, O, I_{\texttt{one-of}}, G\rangle$ where $I_{\texttt{one-of}}$ is obtained from I be replacing every* `one-of`$(\ell_1, \ldots, \ell_n)$ *in I by* `or`$(\ell_1, \ldots, \ell_n)$.

Proposition 5 indicates that if $P_{\texttt{one-of}}$ has a solution then it is also a solution of P. However, this cannot be applied arbitrarily as shown in the following example.

Example 1. *Let $P = \langle\{f, g, h\}, \{a\}, I, G\rangle$ where I = \{*`one-of`(f, g, h), `one-of`$(\neg f, \neg g)$*\}, $a : \neg h \rightarrow g$, and $G = \{g\}$. Intuitively, I specifies that $\neg h$ is true in the initial state as the set of states satisfying I contains only two states $\{f, \neg g, \neg h\}$ and $\{\neg f, g, \neg h\}$. As such, a solution for this problem is a. It is easy to see that replacing the* `one-of` *statements by* `or` *statements indiscreetly leads to a new problem that has no solution.*

The above example shows that `one-of` relaxation is incomplete. Thus, it is critical to find conditions under which the technique can be used and the completeness of the planner is guaranteed. To this end, we formalize the notion of independence of `one-of` statements. For a `one-of`/`or` statement δ, by $lit(\delta)$ we denote the set of literals occurring in δ.

Definition 9. *A* `one-of` *statement δ is* independent *of*

- *a set of literal S if $lit(\delta) \cap S = \emptyset$ and $\overline{lit(\delta)} \cap S = \emptyset$;*
- *an* `one-of` *statement (resp.* `or` *statement) γ if $lit(\delta) \cap lit(\gamma) = \emptyset$ and $\overline{lit(\delta)} \cap lit(\gamma) = \emptyset$.*
- *an initial state description I, where $\delta \in I$, if δ is independent of every element in $I \setminus \{\delta\}$.*

Intuitively, the independency of an `one-of` statement δ of I indicates that the literals occurring in δ do not appear in any other statements of I. We also need the next definition.

Definition 10. *Let a be an action and δ be an* `one-of` *statement δ, we say*

- a maintains δ *if one* $|pre(a) \cap lit(\delta)| \leq 1$ *and for every state s, if* $s \models \delta$ *then* $\Phi(a, s) \models \delta$.
- a negated δ *if* $|pre(a) \cap lit(\delta)| \leq 1$ *and for every state s, if* $s \models \delta$ *then* $\Phi(a, s) \models \bar{\ell}$ *for every* $\ell \in \delta$.

Intuitively, an action maintains an `one-of` statement δ if its execution does not change the overall dependency between the literals in δ. In the benchmarks, this type of actions frequently occurs. For instance, the action of *driving* a truck from one location to another location maintains the `one-of` statement about the locations of the truck; it also maintains the `one-of` statements about the locations of other trucks, as it does not affect the location of other trucks. To verify the property of maintainability between actions and `one-of` statements, we make use of the following lemma.

Lemma 2. *Let* $\delta =$ `one-of`$(p_1, \ldots, p_n)$ *and a be an action with* $C_a = \{\psi_1 \rightarrow \ell_1, \ldots, \psi_k \rightarrow \ell_k\}$. *Let* $\gamma = \{p_1, \ldots, p_n\}$ *and* $L = \{\ell_1, \ldots, \ell_k\}$. *a maintains* δ *if either* $L \cap \gamma = L \cap \bar{\gamma} = \emptyset$*; or the following conditions are satisfied:*

- ○ $|L \cap \gamma| \leq 1$*; and*
- ○ *if* $\ell_i \in (L \cap \gamma)$ *then* $\gamma \cap (pre(a) \cup \psi_i) \neq \emptyset$ *and* $\psi_i \rightarrow \bar{\ell}$ *for every* $\ell \in \gamma \cap (pre(a) \cup \psi_i)$.

The above lemma allows us to check, syntactically, the maintainability of an action and `one-of` statements. A similar lemma can be developed for checking whether an action negates an `one-of` statement. We can now state a completeness of the problem $P_{\texttt{one-of}}$.

Theorem 2. *Let P be the planning problem* $\langle F, O, I, G\rangle$. *If*

- *for every* `one-of` *statement* δ *in I,* δ *is independent of I;*
- *for every action a in O and* `one-of` *statement* δ *in I, a maintains* δ *or a negates* δ*; and*
- *G is a conjunction of literals*

then every solution of P is also a solution of $P_{\texttt{one-of}}$.

The third condition is important. To see why, consider the statement `one-of`(f, g) which satisfies $G = (f \vee g) \wedge (\neg f \vee \neg g)$. It is easy to check that `or`(f, g) does not satisfy G.

We observe that many benchmarks in the literature satisfy the conditions of Theorem 2. For example, the *coins, dispose* domains from the recent IPCs satisfy these property.

CNF— A Planner with CNF-States

We now describe CNF, a conformant planner which uses CNF-states as the representation of belief states. The planner is a modification of DNF. Like other heuristic search based planners, CNF implements a best first search algorithm with the following specifics: each node in the search is a pair of a CNF-state φ and the plan p such that $\varphi = \widehat{\Phi_{CNF}}(p, \varphi_I)$ where φ_I is the initial CNF-state. We omit the search algorithm for space consideration. In what follows, we describe some important considerations, which are different from other planners and contribute to the good performance of CNF.

Computing Successor CNF-States

Given a CNF-state φ and an action a with the set of conditional effects C_a, computing the successor CNF-state $\varphi' = \Phi_{CNF}(a, \varphi)$, by Definition 7, includes the following steps:

1. If $pre(a)$ is not true in φ then φ' is undefined.
2. If $pre(a)$ is true in φ, then φ' is computed in three steps:
   (a) $enb_a(\varphi)$
   (b) $\Psi = \{\mathit{Update}(\phi, e(a, \phi)) | \phi \in enb_a(\varphi)\}$
   (c) $\varphi' = \times[\Psi]$

The algorithms for computing $\mathit{Update}(\phi, e(a, \phi))$ and $\times[\Psi]$ are quite straightforward following Def. 4 (for *Update*), and Def. 3 (for $\times$) so we omit them here. For the simplification of the computation, we implement a preprocessor which combines all effects with the same antecedent into one of the form $\psi \rightarrow \eta$, where η is a set of literals. The algorithm for computing $enb_a(\varphi)$ is given in Alg. 1. Observe that it also computes $e(a, \phi)$ for each $\phi \in enb_a(\varphi)$, eliminating some redundant computations. More precisely, the computation in line 7 is done as follows. If $\phi \models \psi$ then $Z = \{\phi\}$ and $e(a, \phi) = e(a, \phi) \cup \eta$; If $\phi \models \neg\psi$ then $Z = \{\phi\}$ and $e(a, \phi)$ does not change; otherwise, $Z = \{\phi_1, \phi_2\}$ where $\phi_1 = r(\phi \wedge \psi)$ and $\phi_2 = r(\phi \wedge \neg\psi)$ with $e(a, \phi_1) = e(a, \phi) \cup \eta$ and $e(a, \phi_2) = e(a, \phi)$.

Algorithm 1 Computing $enb_a(\varphi)$

```
1:  Input: CNF state $\varphi$, action $a$
2:  Output: $enb_a(\varphi)$, $e(a, \phi)$ for $\phi \in enb_a(\varphi)$
3:  Let $X = \{\varphi\}$, $e(a, \phi) = \emptyset$ for $\phi \in enb_a(\varphi)$
4:  for each effect $\psi \rightarrow \eta$ in $C_a$ do
5:      Let $Y = \emptyset$
6:      for each $\phi \in X$ do
7:          compute $Z = \phi + \psi$, update $e(a, \phi')$ for $\phi' \in Z$
8:          $Y = Y \cup Z$
9:      end for
10:     Set X = Y
11: end for
12: return $X$
```

For computing $\times[\Psi]$, we use the following proposition to reduce the amount of computation

Proposition 6. *Let* φ *and* ψ *be two CNF formulae, c be a clause in* φ*, and* c' *be a clause in* ψ*. It holds that*

- *If* $c = c'$ *then* $r(\varphi \times \psi) = r((\varphi \setminus c) \times (\psi \setminus c') \cup c)$
- *If* $c \subset c'$ *then* $r(\varphi \times \psi) = r(\varphi \times (\psi \setminus c') \cup c')$

Observe that if $|\varphi| = n$ (φ has n clauses) and $|\psi| = m$ then the first item says that the size of the cross-product can be reduced by $n + m - 1$ clauses. The second item indicates that it can be reduced by $n - 1$ clauses. Our experiments show that this contributes to a significant reduction of search time, ranging between 20% and 90% in the benchmark problems. The main reason is that, without using optimization of Prop. 6, CNF spends most of search time for computing cross-product of CNF-formulae which, if in the same CNF-belief state, usually contains many common clauses since they are originated from the same CNF-formula in the transformation to enabling form. In addition, the *Update* function creates unit clause(s) which can subsume many other clauses in the CNF-states of the same CNF-belief state.

Checking Entailment

Checking whether a CNF-state satisfies a set of literals is needed whenever the planner needs to check for the executability condition of an action and the goal satisfaction of CNF-states. It is also needed during the computation of the enabling CNF-belief states. In general, checking whether a literal ℓ (resp. set, clause) is true or false in a CNF-state φ

needs to a call to a SAT-solver. Thus, reducing the number of calls to the SAT-solver is important for the performance of CNF. The representation of belief states as CNF-states provides an easy way to check for entailment in several cases. CNF takes advantages of this fact and reduces the number of calls to the SAT-solver by implementing the following tests before a call to the SAT-solver is made: (*i*) if $\ell \notin \varphi$ then $\varphi \not\models \ell$; (*ii*) if φ contains a clause c' that subsumes a clause c then $\varphi \models c$. With this simplifications, the running time portion of the SAT-solver is manageable for most cases. In our experiment, we observe that it is always less than 3%.

Heuristic

Since CNF is a modification of DNF, it uses the number of satisfied subgoals to guide its search. However, it does not consider the equivalence of the cardinality heuristic. The reasons for this decision are twofold. By using the number of satisfied subgoals, we can evaluate the impact of the belief state representation (using CNF-states) when we compare the performance of CNF and DNF. CNF does not use the cardinality heuristic because of the less the clauses in a CNF-state, the more uncertainty it contains, in general.

Preprocessing Input

CNF is developed by modifying the source code of DNF and CPA. As a result, CNF accepts the same input as DNF and incurs the overhead generated by the preprocessing phase as does DNF. However, CNF does not make use of `one-of` combination or goal-splitting techniques.

`one-of` Relaxation

CNF makes use of the `one-of` relaxation technique described in the previous section.

Experimental Evaluation

We evaluate CNF against the following conformant planners: DNF, CPA, CFF, POND, and `t0` using conformant planning benchmarks found in the literature. To the best of our knowledge, these planners currently yield the best performance in these domains. We obtained DNF, CPA, and `t0`, from their respective authors. CFF was downloaded from its author website. We use POND version 1.1.1 with default setting.

All the experiments have been carried out using a set of three dedicated Linux workstations of the same configuration: Intel Core 2 Dual 9400 2.66GHz 4GB RAM. The timeout limit is set to 2 hours. The results of our experiments are divided into different tables. In Tables 1, 3, and 4, each column contains the performance of a planner. We report the time and plan length for each planner. 'OM', 'TO', and 'F' denotes out of memory, time out, or abnormal termination of the planner. Due to space limitation, we only report a few large instances of each benchmark and do not report the number of states that are generated and explored by each planner. This number is included in the full version of this paper. In the following, we discuss each table and evaluate the strength and weakness of CNF against other planners.

Benchmarks From The Literature

Table 1 contains the results obtained from experiments on the domains *block*, *raokeys* (raok-n), and *uts-cycle* (uts-c-n) from IPC-2008. *sortnet* is from IPC-2006. *bomb* and *gripper* are from the author of CFF. The remaining problems including *corner-cube* (cc-n-m), *look-and-grab* (lng-n-m-k), and *sort-number* (sortnum-n) are from the package of `t0`.

The problems in Table 1 share the common property that they do not contain an `one-of` statement which satisfies the relaxation condition. Therefore, CNF can not take the advantage of relaxation technique for these problems. The results show that CNF and DNF are the best on the *bomb, corner-cube, grip, look-and-grab* domains and the performance of these two planners are almost the same on these domains. However, CNF scales up better on *corner-cube* while DNF is better on *look-and-grab*. The execution time and length of plans found by CNF and DNF are the same for *bomb* and *gripper*. For *sort-number*, `t0` is the best on most instances but only CNF is able to deal with the largest instance. We observe that CNF spends more time in transforming CNF-belief states to CNF-states in this problem while the size of the representation of DNF increases very fast. POND outperforms the others on *block* and *sortnet* but its performance is pretty poor on the other domains. CFF is very good at *gripper*, but it does not scale up well as CNF or DNF does on the larger instances. `t0` is the best on *raokeys*. This planner also performs very well on many small problem instances.

Observe that although DNF outperforms all the other planners on small instances of the *corner-cube* domain but the time increases by a faster rate comparing to CNF. This depends on the number of states generated and explored by DNF and CNF: $24969/5717$ v.s. $18321/6362$ for *corner-cube-40-20* and $300287/62027$ v.s. $115330/39283$ for *corner-cube-119-59*, respectively.

On the other hand, CNF does not do so good in the *uts-cycle* domain due to the explosion of the CNF-state size as shown in Table 5. The overhead of the function $\times$ of converting a CNF-belief state to the equivalent CNF-state is another reason. For example, for the instance uts-cycle-3, the real search time (excluding the input theory translation time) is 3.325 seconds. Whereas, the execution time spent on the function $\times$ is 3.237, i.e. most of the search time. We present the overhead incurred by CNF for some domains in Table 2. In this table, t_trans, t_search, and t_CNF are the time used by the preprocessing phase (which DNF does also use), the time for searching for a solution, and the time incurs by the transformation of CNF-belief states to CNF-states.

Challenging Problems

Table 3 contains the experimental results with domains that are challenging for current state-of-the-art planners. Among these, *coins* and *dispose* (ds-m-n) are from the IPC-2006 and IPC-2008. The others are from the challenging domains proposed in (Palacios and Geffner 2007) including *1-dispose* (1d-n-m) and *push* (push-n-m). Indeed, only few planners can solve these problems and none can scale up when the number of objects/constants become large. For coins-21 and larger instances, the main difficulty, which makes these problems challenging, is the huge size of the belief states encountered during the search or the huge size of the initial belief state.

The domains in Table 3 fall in the class of problems where

Problem	CNF	DNF	CPA	T0	CFF	POND
block-1	0.56/7	0.67/7	0.68/4	0.08/5	0.02/6	**0.01**/6
block-2	0.83/18	0.68/38	0.76/14	0.2/23	TO	**0.06**/34
block-3	TO	215/331	OM	48/80	TO	**3.9**/80
bomb-20-10	0.74/30	0.77/30	3.6/24	F	**0.05**/30	OM
bomb-50-10	1.28/90	1.28/90	21/58	F	**1.16**/90	OM
bomb-100-10	**2.7**/190	**2.7**/190	110/110	F	34/190	OM
bomb-100-20	5.3/180	**5.2**/180	244/118	F	28/180	OM
cc-20-10	2.37/138	**2.07**/217	F	2.1/258	2.5k/332	TO
cc-40-20	13.4/343	**7.75**/535	F	87/918	TO	TO
cc-64-32	49.5/1106	**21.2**/872	F	F	TO	TO
cc-99-49	90.6/1542	**89.2**/1553	F	F	TO	TO
cc-119-59	**180**/2230	229/3522	F	F	TO	TO
gripper-30	3.3/144	3.7/144	17/72	11/118	**0.5**/174	TO
gripper-50	8.6/234	8.7/234	106/106	52/198	**3**/294	TO
gripper-80	22.2/350	22.5/350	432/166	234/318	**18**/474	TO
gripper-100	31.9/438	**31.2**/438	970/206	421/398	>43	TO
lng-4-1-2	1.21/16	**1.06**/18	1.17/26	27/12	83/105	OM
lng-4-2-2	**1.22**/4	1.33/4	1.38/4	31/4	TO	OM
lng-4-2-3	**2.8**/4	3.13/4	3.91/4	141/4	TO	OM
lng-4-3-3	**3.01**/4	3.53/4	4.48/4	F	TO	OM
lng-7-2-2	26.4/58	**17.9**/48	21/28	F	TO	OM
lng-7-3-2	22.9/14	**20.8**/14	24.7/12	F	TO	OM
raok-2	1.65/27	0.57/26	1.1/32	**0.04**/21	0.07/34	F
raok-3	2754/919	1.68/153	3.8/152	**0.25**/66	12/102	F
raok-4	TO	TO	TO	F	TO	F
sortnet-5	1.2/15	0.94/15	0.94/12	0.26/15	NA	**0/12**
sortnet-10	21.9/55	1.85/54	3.18/39	OM	NA	**0.03/38**
sortnet-15	257/119	35.9/118	244/65	F	NA	**0.14/65**
sortnum-5	1.56/10	2.57/10	OM	1.92/10	2.9/10	**0.49**/10
sortnum-6	22.8/15	397/15	OM	**18.1**/15	TO	19.4/15
sortnum-7	539/21	OM	OM	**91**/21	TO	1077/21
sortnum-8	**14277**/28	OM	OM	F	TO	TO
uts-c-3	4.22/3	0.54/3	1.19/3	**0.15**/3	NA	F
uts-c-4	TO	0.56/6	18.3/6	**0.47**/7	NA	F
uts-c-5	TO	**0.73**/10	OM	1.8/10	NA	F

Table 1: Benchmarks from Literature

the `one-of` relaxation is applicable. As we can see, CNF outperforms other planners in many challenging domains (*dispose, 1-dispose, and push*) and shows great performance in the *coins* domain. In fact, only CNF was able to solve the instances (-21 to -30) of the *coins* domain. In smaller instances, CNF and DNF are comparable but the high memory usage of DNF results in the "OM" error of DNF in large instances.

New Challenging Problems

In this subsection, we introduce a new set of problems that play to the advantage of CNF. They are variants of the challenging problems of the problems in Table 3. Instead of `one-of` statements, we use the `or` statements. To illustrate the modification, let us use the *dispose* problem.

In this problem, there are m different objects $o_1, \ldots, o_m$

Problem	t_trans	t_search	t_CNF	Problem	t_trans	t_search	t_CNF
block-1	0.53	0.029	0.013	b-50-10	1.01	0.272	0.032
gripper-30	1.19	2.11	0.23	c-20-40	0.75	1.62	0.83
sortnet-5	0.85	0.35	0.15	lng-4-1-2	0.96	0.25	0.11
uts-c-3	0.895	3.325	3.237	sortnum-5	0.613	0.947	0.682

Table 2: Time Distribution: Search vs. Translation vs. Transformation in CNF

Problem	CNF	DNF	CPA	T0	CFF	POND
coins-10	0.6/35	0.62/27	0.9/36	**0.04/26**	0.12/38	0.5/46
coins-15	0.91/81	0.97/67	7/362	**0.12/79**	2.6/89	10/124
coins-20	1.15/114	1.41/99	17/105	**0.15/107**	16/143	105/153
coins-21	**1478/7321**	OM	OM	F	TO	TO
coins-25	**195/1843**	OM	OM	F	TO	TO
coins-29	**999/5368**	OM	OM	F	TO	TO
ds-4-3	0.88/93	1.2/185	4.9/92	**0.2**/122	0.6/73	55/125
ds-4-5	1.15/155	1.7/180	11.3/126	**0.8**/145	2.5/107	TO
ds-8-3	**27.5/392**	37.4/447	778/413	133/761	TO	OM
ds-8-5	**32/537**	65/878	2152/541	F	TO	OM
ds-10-3	**158/531**	193/680	4694/648	2388/1360	TO	OM
ds-10-5	**175/840**	274/1286	TO	F	TO	TO
ds-10-9	**222/1543**	787/2178	TO	F	TO	TO
1d-3-2	**0.74**/60	0.78/36	0.76/27	0.9/38	TO	5.8/54
1d-3-3	**1.04**/60	1.86/36	1.87/27	48/36	TO	77/50
1d-3-4	**1.67**/60	12.5/36	14.3/27	F	TO	OM
1d-6-2	**10.8**/492	15.4/186	33.7/124	F	TO	OM
1d-6-3	**18.1**/492	506/186	OM	F	TO	OM
1d-6-4	**30.4**/492	OM	OM	F	TO	OM
1d-10-4	**667**/3806	OM	OM	F	TO	OM
1d-10-5	**1240**/3806	OM	OM	F	TO	OM
push-4-1	0.94/41	1.17/41	1.61/51	**0.16**/78	0.28/46	7.3/65
push-4-3	1.06/105	2.1/194	144/847	**0.66**/164	1.3/48	TO
push-4-5	**1.45**/145	5.05/331	OM	2.04/260	126/50	OM
push-8-1	**25.8/163**	27.1/163	29.6/169	63/464	TO	TO
push-8-2	**29.6/396**	41.2/903	4254/429	F	TO	TO
push-8-3	**31.9/303**	74/1477	OM	F	TO	TO
push10-6	**227/1194**	2235/6382	OM	F	TO	TO
push10-7	**259/1221**	OM	OM	F	TO	TO

Table 3: Challenging Domains

and n different places $p_1, \ldots, p_n$. The goal is to find a plan for a robot to pickup the objects and drop them at a place where a trash can is available. Initially, we don't have any information about which object at which place. So, in the initial information there are m `one-of` clauses of following form: `one-of`$(at(o_i, p_1), \ldots, at(o_i, p_n))$ for $i = 1, \ldots, m$. The robot can move between places by the actions up, $down$, $left$, and $right$; pickup objects at different locations by the action $pickup(o_i, p_j)$; hold an object $holding(o_i)$.

The new problem, denoted by *or-dispose*, is as follows. In this new problem, each object is associated with a type, i.e., each o_i is an object type, e.g. pens, books ... The predicate $at(o_i, p_j)$ now means "there exists an object of type o_i at place p_j." Similarly, $holding(o_i)$ means "holding some object(s) of type o_i." The action $pickup(o_i, p_j)$ allows the robot to pickup all the objects of type o_i at place p_j. Suppose, initially, all we know about the objects is that for each object type o_i, there is at least one instance of o_i lying in some places p_j's. In effect, we replace the `one-of` statement `one-of`$(at(o_i, p_1), \ldots, at(o_i, p_n))$ by the `or` statement `or`$(at(o_i, p_1), \ldots, at(o_i, p_n))$ in the domain specification of *dispose*.

Observe that, when `one-of` relaxation is applied, the description of the initial state of *dispose* becomes exactly the same as that of *or-dispose*. Hence, the size of the CNF state representing the initial belief state does not change from *dispose* to *or-dispose* if the `one-of` relaxation is applied. As

Problem	CNF	DNF	CPA	T0	CFF	POND
or-coins-10	0.65/35	OM	OM	**0.032/26**	0.11/38	0.5/46
or-coins-20	1.1/114	OM	OM	**0.15/107**	15.9/143	86/121
or-coins-21	**1486**/7321	OM	OM	F	TO	TO
or-ds-4-3	0.86/93	81/117	OM	**0.26**/122	0.46/73	63.6/126
or-ds-4-5	**1.15**/155	OM	OM	**1.15**/145	1.9/107	OM
or-ds-8-3	**27.85/392**	OM	OM	278/761	TO	OM
or-ds-8-5	**31.9/537**	OM	OM	F	TO	OM
or-ds-10-9	**222/1543**	OM	OM	F	TO	OM
or-1d-3-2	**0.74**/60	1.15/32	OM	F	TO	6.17/50
or-1d-3-4	**1.67**/60	420/32	OM	F	TO	OM
or-1d-6-2	**10.8**/492	190/150	OM	F	TO	OM
or-1d-6-3	**18.1**/492	OM	OM	F	TO	OM
or-1d-10-5	**1240**/3806	OM	OM	F	TO	OM
or-push-4-1	0.94/41	0.83/61	52/49	0.38/78	**0.26**/46	1.86/65
or-push-4-3	1.06/105	115/147	OM	11/189	**0.8/48**	125/120
or-push-8-1	**26.6**/163	27.2/134	OM	2.8k/367	TO	TO
or-push-8-2	**29.8**/396	OM	OM	TO	TO	TO
or-push-10-7	**259**/1221	OM	OM	TO	TO	TO

Table 4: New Challenging Domains

such, CNF will be able to solve the new problems without difficulty.

The new set of problems contains also the *or*-version of other problems such as *or-coins*, *or-dispose*, *or-1-dispose*, and *or-push*.

Table 4 displays the results of all planners on the set of new problems. Due to the extremely large size of disjunctive normal form formulae representing belief states in these problems, a DNF belief state representation based planner, e.g. DNF and CPA, appears to have very poor performance on these problems. In these problems, the `one-of` combination technique is not applicable and therefore DNF and CPA perform poorly on these problems. On the contrary, due to the capability of maintaining a compact size of CNF formulae representing belief states, CNF outperforms impressively not only DNF belief state representation based planners, but also all other competitive state-of-the-art conformant planners. It is worth mentioning that there is no significant difference between the performance of any other planner on these problems in comparison with that on the original problems.

Size of CNF-states

We define the size of a CNF-state the total of the size of every non-unit clause in the CNF-state. The experiments reveal that the size of successor CNF-states keep decreasing in most problems. The reason is that updating a CNF-state by a literal ℓ removes all the clauses containing ℓ or $\bar{\ell}$. On the other hand, most clauses in $(\varphi_\ell - \ell) \times (\varphi_{\bar{\ell}} - \bar{\ell})$ are usually trivial or subsumed by another clause in the formula. Tables 5 and 6 report the size of the initial CNF-state and the average size of the generated CNF-states for several instances. Table 6 also shows the effectiveness of the `one-of` relaxation by additionally reporting those sizes for some problems in Table 3 when the `one-of` relaxation is not used.

Problem	Init./Avg.	Problem	Init./Avg.	Problem	Init./Avg.
block-1	47/39	cc-99-49	12/6	sortnet-10	0/43
block-2	105/145	cc-119-59	12/6	sortnet-15	0/87
bomb-m-n	0/0	lng-4-3-3	768/57	sortnum-5	0/28
uts-c-3	132/2820	lng-7-3-2	4802/134	sortnum-7	0/102

Table 5: Size of CNF-states: Initial vs. Average

Problem	Relaxation	No Relax.	Probem	Relaxation	No Relax.
coins-20	54/28	402/200	ds-8-5	320/121	20k/4.7k
coins-22	170/45	1.7k/1.3k	ds-10-3	300/99	30k/5.7k
1d-3-3	27/26	243/446	push-4-5	80/22	1280/165
1d-6-2	72/25	2592/511	push-8-3	192/64	12k/203

Table 6: Size of CNF-states: Applying Relax. vs. Not Applying

Conclusion and Future Work

In this paper, we introduce a new conformant planner, CNF, which uses a special type of CNF formulae to represent belief states. The overall performance of CNF on several benchmark problems shows that it is competitive with other state-of-the-art conformant planners in many domains. More importantly, the new planner provides a better scalability than all other planners; it can solve larger instances of many problems that were challenging to previous conformant planners. We propose a new technique, called `one-of` relaxation, aimed at reducing the size of the initial belief state. We also propose a new set of benchmarks that provides new challenges to conformant planners.

Although CNF displays good performance, it does have some weaknesses which need to be addressed. First of all, the overhead incurred in the computation of the $\times$ function is sometimes larger than the actual time spent for searching for a solution. Furthermore, an improved heuristic is also desirable. Finally, the performance of CNF and DNF suggests that a combination of the two techniques into one planner might yield interesting result.

References

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In *ICAPS*, 355–364.

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.

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

Palacios, H., and Geffner, H. 2007. From Conformant into Classical Planning: Efficient Translations that may be Complete Too. In *ICAPS*.

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

To, S. T.; Pontelli, E.; and Son, T. C. 2009. A Conformant Planner with Explicit Disjunctive Representation of Belief States. In *Proceedings of the 19th International Conference on Planning and Scheduling (ICAPS)*.

Tran, D.-V.; Nguyen, H.-K.; Pontelli, E.; and Son, T. C. 2009. Improving performance of conformant planners: Static analysis of declarative planning domain specifications. In *PADL*, LNCS 5418, 239–253. Springer.