Contingent Planning as AND/OR Forward Search with Disjunctive Representation*

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

This paper introduces a highly competitive contingent planner, that uses the novel idea of encoding belief states as disjunctive normal form formulae (To et al. 2009), for the search for solutions in the belief state space. In (To et al. 2009), a complete transition function for computing successor belief states in the presence of incomplete information has been defined. This work extends the function to handle *non-deterministic* and *sensing* actions in the AND/OR forward search paradigm for contingent planning solutions. The function allows one, under reasonable assumptions, to compute successor belief states efficiently, i.e., in polynomial time. The paper also presents a novel variant of an AND/OR search algorithm, called PrAO (Pruning AND/OR search), which allows the planner to significantly prune the search space; furthermore, by the time a solution is found, the remaining search graph is also the solution tree for the contingent planing problem. The strength of these techniques is confirmed by the empirical results obtained from a large set of benchmarks available in the literature.

Introduction

Contingent planning (Peot and Smith 1992) is the problem of finding conditional plans given incomplete knowledge about the initial world and uncertain action effects. The contingent plan allows the agent to act, at execution time, conditionally depending on the observed values of some uncertain properties of the world; and guarantees to achieve the goal no matter what the actual initial world the agent starts from and which actual action effects occur. Contingent planning is one of the hardest problems considered in the area (Baral *et al.* 2000; Rintanen 2004).

One of the best-known and successful approaches to contingent planning is to transform the problem into an AND/OR search problem in the belief state space. Using the notion of *belief state* is convenient for capturing the semantic of the uncertain world and defining the transition function for computing the successor (uncertain) world state. Yet it is impractical to use belief states themselves in the implementation of a planner due to their exponential size. The question is then how to represent belief states and, given a representation, how to define a transition function for computing successor belief states under conditional action effects. To address this, (Bertoli et al. 2001) proposed the use of *binary decision diagrams* (BDDs) (Bryant 1992) to represent belief states in a model checking based planner, called MBP. Later, (Bryce *et al.* 2006) used BDDs to represent literals and actions in the planning graph for computation of heuristics used to search for solutions in their contingent planner, called POND. The use of the BDD representation is advantageous since it is more compact than the belief state itself and it allows one to check whether a literal holds in a world state easily. Nevertheless, the size of a BDD representation is still very large and sensitive to the order of the variables. Moreover, computing successor belief states in BDDs form during the search is very expensive, requiring the generation of intermediate formulae of exponential size. This explains why MBP and POND do not scale well as shown in (Hoffmann and Brafman 2005; Albore et al. 2009).

At the other extreme of belief state representation, the proposal in (Brafman and Hoffmann 2004; Hoffmann and Brafman 2005) represents belief states implicitly through the action sequences that lead to them from the initial belief state, and uses forward search in the belief space for solutions. The advantage of the representation they use is easily seen, as it requires very little memory, scaling up pretty well on a number of problems. The trade-off is that it incurs a great amount of repeated computation. Especially, checking whether a proposition holds after the execution of even one single action in the presence of incomplete information is co-NP complete. We believe that this is one of the reasons for their planners to hardly find a solution for even small instances of harder problems, where the structure of the actions is complex or there exist unknown propositions in the preconditions of the conditional effects as observed in (Albore et al. 2009; To et al. 2010).

(Albore et al. 2009) proposed a different approach to contingent planning by translating a contingent problem into an AND/OR search problem in the state space whose literals represent the beliefs over the original problem, assuming that the uncertainty lies in the initial world only and all actions are deterministic. They showed that the translation is polynomial under some assumptions. This approach is more efficient, i.e., the planners can solve harder problems of larger size, compared to the previous approaches. The

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

disadvantage is that the number of literals in the translated problem can be exponential in the number of literals in the original problem, making the state space extremely large and prohibiting the planners to scale up.

Recently, we proposed a new approach to conformant planning by using a special, compact form of disjunctive formulae, called *minimal DNF*, to represent belief states and defining a direct complete transition function which allows computing the successor belief states encoded in this representation in the presence of incomplete information efficiently, i.e., polynomial under reasonable assumptions (To et al. 2009). The advantage of this method is confirmed by the fact that our planner DNF can solve much larger instances of many benchmarks, including the hardest problems given in the literature. The performance of DNF is not as good on the problems where the size of disjunctive formulae encoding the belief states is too large even in a very compact (disjunctive) form. To address this, we proposed some compact conjunctive normal form (CNF) formulae and prime implicate formulae as other representations of belief states (To et al. 2010; 2010b).

This work proposes a new approach to contingent planning that uses the minimal-DNF representation introduced in (To et al. 2009). To this end, we extend the transition function for computing the successor belief states in the presence of incomplete information defined in (To et al. 2009) to also handle uncertain action effects and sensing actions. On the other hand, we develop a novel variant of AND/OR forward search algorithms, called PrAO, for contingent planning. Key to PrAO is a novel safe pruning technique which can significantly reduce the search space in many cases. Furthermore, solution extraction in PrAO is fairly simple, as the remaining search graph, by the time a solution is found, is also the solution tree due to the pruning technique. We implement the algorithm in a planner, called DNF$_{ct}$, and compare DNF$_{ct}$ with other state-of-the-art contingent planners. The experimental results validate the approach of this paper.

The paper is organized as follows. We next review the background of contingent planning and minimal-DNF representation of belief states. We then present PrAO, an AND/OR search algorithm with a new pruning technique implemented in DNF$_{ct}$ and the experimental evaluation of DNF$_{ct}$. We conclude with the directions of future work.

Background: Contingent Planning

The *contingent planning problem* is defined as a tuple $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$, 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 $\neg p$. $\bar{\ell}$ denotes the complement of a literal ℓ—i.e., $\bar{\ell} = \neg\ell$, where $\neg\neg p{=}p$ for $p{\in}F$. For a set of literals L, $\overline{L} = \{\bar{\ell} \mid \ell \in L\}$. A conjunction of literals is often represented as the set of its conjuncts.

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 precondition of action a and $O(a)$ is a set of action outcomes. Each outcome $o(a)$ in $O(a)$ is a set of conditional effect of the form $\psi \rightarrow \ell$ (also written as $o(a) : \psi \rightarrow \ell$), where ψ is a set of literals and ℓ is a literal. Each observation ω in Ω is a tuple $\langle pre(\omega), \ell(\omega)\rangle$, where $pre(\omega)$ is the precondition of ω which is a set of literals, and $\ell(\omega)$ 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 $O(a)$ occur. However, which outcome that occurs is uncertain.

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 $o(a)$ is

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

Let $\Phi(o(a), s) = s \setminus \overline{e(o(a), s)} \cup e(o(a), s)$. The transition function that maps pairs of actions and belief states into a belief state in the planning domain of $\mathcal{P}$ is defined by $\Phi(a, S) = \{\Phi(o(a), s) \mid s \in S, o(a) \in O(a)\}$ if $S \neq \emptyset$ and $S \models pre(a)$; $\Phi(a, S) = undefined$, otherwise.

Example 1. *Given a domain $F = \{head\}$, a belief state S that contains one single state $S = \{\{head\}\}$, an action $flip$ with $pre(flip) = \{true\}$ and $O(flip) = \{o_1, o_2\}$, where o_1 contains one effect $o_1 : true \rightarrow head$ and o_2 contains another effect $o_2 : true \rightarrow \neg head$. Then one can easily compute: $\Phi(o_1, \{head\}) = \{head\}$, and $\Phi(o_2, \{head\}) = \{\neg head\}$. Hence, $\Phi(flip, S) = \{\{head\}, \{\neg head\}\}$.*

Observe that in Example 1, the non-deterministic action $flip$ causes the certain belief state S to become uncertain.

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

Given a contingent planning problem $\mathcal{P}$, a structure T constructed from the actions and observations of $\mathcal{P}$ is said to be a *transition tree* of $\mathcal{P}$ 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 $\perp$. The result of the execution of a transition tree T in a belief state S, denoted by $\widehat{\Phi}(T, S)$, is a set of belief states defined as follows:

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

Note that the definition of $\widehat{\Phi}$ allows the application of an

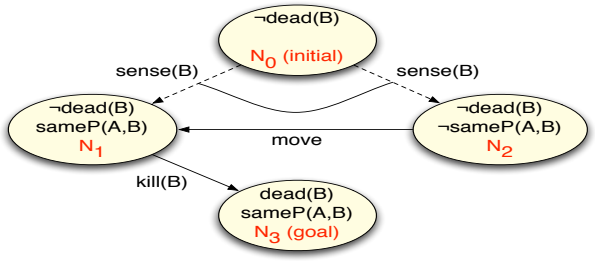

Figure 1: A contingent solution in an AND/OR forward search.

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 a *solution* of $\mathcal{P}$ if T is *finite* and every belief state in $\widehat{\Phi}(T, S_I)$ satisfies the goal G.

Example 2. *Consider a simple problem $\mathcal{P}$ with an agent A and a bug B in a house with two rooms. A can move around and kill B if they are in the same room. A can sense whether B is in the same room or not. The problem $\mathcal{P}$ is given as F=$\{sameP(A,B), dead(B)\}$, A =$\{move,\ kill(B)\}$, Ω=$\{sense(B)\}$, I=$true$, and G=$dead(B)$; where $move$=$(true,\ \{sameP(A,B) \rightarrow \neg sameP(A,B),\ \neg sameP(A,B) \rightarrow sameP(A,B)\})$, $kill(B)$=$(sameP(A,B), \{true \rightarrow dead(B)\})$, and $\ell(sense(B))$=$sameP(A,B)$. A solution tree for $\mathcal{P}$ is:*

"$sense(B)(kill(B) \mid move \circ kill(B))$"$(Figure\ 1)$.

DNF Representation

We review the *minimal-DNF* representation of belief states and present an extension of the transition function provided in (To et al. 2009) which also handles non-deterministic actions and observations, required in contingent planning, in addition to incomplete information.

A *partial state* is a consistent set of literals. A state s is a *completion* of a partial state δ if $\delta \subseteq s$. The *extension* of δ, denoted by $ext(\delta)$, is the set of all completions of δ.

A *DNF-state* is a set of partial states that does not contain a pair of δ_1 and δ_2 such that $\delta_1 \subset \delta_2$.

Let Δ be a DNF-state. Let $\min(\Delta) = \Delta \setminus \{\delta \mid \exists \delta' \in \Delta . \delta' \subset \delta\}$. $\min(\Delta)$ is a DNF-state equivalent to Δ. Let $ext(\Delta) = \bigcup_{\delta \in \Delta} ext(\delta)$ be the *extension* of Δ. $ext(\Delta)$ is a belief state equivalent to Δ.

Given a partial state δ, a literal ℓ is true (resp. false) in δ if $\ell \in \delta$ (resp. $\bar{\ell} \in \delta$), denoted by $\delta \models \ell$ (resp. $\delta \models \bar{\ell}$). A literal ℓ is said to be known in δ if it is true or false in δ.

For a set of literals γ, $\delta \models \gamma$ if $\gamma \subseteq \delta$. For a DNF-state Δ and a set of literals γ, $\Delta \models \gamma$ if $\forall \delta \in \Delta . \delta \models \gamma$.

Let δ be a partial state and γ be a consistent set of literals. The *partial extension $\delta + \gamma$ of δ w.r.t. γ* is defined as

$$\delta + \gamma = \begin{cases} \{\delta\} & \text{if } \gamma \subseteq \delta \text{ or } \overline{\gamma} \cap \delta \neq \emptyset \\ \{\delta \cup \gamma\} \cup \{\delta \cup \{\bar{l}\} | l \in \gamma \setminus \delta\} & \text{otherwise} \end{cases} \quad (1)$$

$\delta + \gamma$ is a DNF-state satisfying δ in which γ is known. For a DNF-state Δ, let $\Delta + \gamma = \min(\bigcup_{\delta \in \Delta} (\delta + \gamma))$.

The definition of *enabling* notion is extended as follows

Definition 1. *Let $o(a)$ be an outcome of action a. A partial state δ is called* enabling *for $o(a)$ if for every conditional effect $o(a) : \psi \rightarrow \ell$, either $\delta \models \psi$ or $\delta \models \neg\psi$ holds. A DNF-state Δ is* enabling *for $o(a)$ if every partial state in Δ is enabling for $o(a)$.*

For an outcome $o(a)$ of action a and a partial state δ, let $exp_{o(a)}(\delta) = ((\delta + \psi_1) + \ldots) + \psi_k$ where $o(a) = \{\psi_1 \rightarrow \ell_1, \ldots, \psi_k \rightarrow \ell_k\}$. For a DNF-state Δ, $exp_{o(a)}(\Delta) = \bigcup_{\delta \in \Delta} exp_{o(a)}(\delta)$. One can prove the following:[1]

Proposition 1. *For every partial state δ, DNF-state Δ, and an outcome $o(a)$ of action a:*

- *$exp_{o(a)}(\delta)$ is a DNF-state which is equivalent to δ and enabling for $o(a)$.*
- *If Δ is a DNF-state, then $\exp_{o(a)}(\Delta)$ is a DNF-state equivalent to Δ and enabling for $o(a)$.*

For an action a and a partial state δ, the effect of a in δ if the outcome $o(a)$ occurs, denoted $e(o(a), \delta)$, is defined as $e(o(a), \delta) = \{l \mid \psi \rightarrow l \in o(a), \delta \models \psi\}$. The result of execution a in δ if $o(a)$ occurs is defined by $res(o(a), \delta) = \delta \setminus \overline{e(o(a), \delta)} \cup e(o(a), \delta)$. We are now ready to define the transition function Φ_{DNF} which maps pairs of actions and DNF-states into DNF-states as follows.

Definition 2. *Let Δ be a DNF-state and a be an action. The execution of a in Δ results in a DNF-state, denoted by $\Phi_{DNF}(a, \Delta)$, defined as follows: $\Phi_{DNF}(a, \Delta) = \min(\bigcup_{o(a) \in O(a)} \{res(o(a), \delta') \mid \delta' \in exp_{o(a)}(\Delta)\})$ if $\Delta \models pre(a)$ and $\Phi_{DNF}(a, \Delta) = undefined$ otherwise.*

Proposition 2. *Given a DNF-state Δ and an action a. If $|O(a)|$ is bounded (by a constant); for each $o(a)$ in $O(a)$, $|o(a)|$ is bounded; and for each $\psi \rightarrow l$ in $O(a)$, $|\psi|$ is bounded. Then $\Phi_{DNF}(a, \Delta)$ is polynomial in $|\Delta|$.*

When executing a sensing action ω in a DNF-state Δ, Δ is split into two DNF-states, one satisfies $\ell(\omega)$ and the other satisfies $\overline{\ell(\omega)}$ and their union set is equivalent to Δ. To do so, we need first to extend Δ w.r.t. $\ell(\omega)$ so that $\ell(\omega)$ is known in each partial state in the new DNF-state. Application of an observation ω in a DNF-state Δ results in the following two DNF-states: $\Delta^+_\omega = \{\delta \mid \delta \in \Delta + \{\ell(\omega)\} \wedge \delta \models \ell(\omega)\}$ and $\Delta^-_\omega = \{\delta \mid \delta \in \Delta + \{\ell(\omega)\} \wedge \delta \models \overline{\ell(\omega)}\}$.

Proposition 3. *If Δ is a DNF-state and ω is an observation then Δ^+_ω and Δ^-_ω are DNF-states.*

Let $\widehat{\Phi_{DNF}}$ be the extended transition function that maps a pair composed of a transition tree and a DNF-state to a set of DNF-states, defined in the same manner as $\widehat{\Phi}$ is, where Φ is replaced with Φ_{DNF} and the belief state S is replaced with the DNF-State Δ. The next theorem shows that $\widehat{\Phi_{DNF}}$ is equivalent to the complete semantics defined by $\widehat{\Phi}$.

Theorem 1. *Let Δ be a DNF-state and T be a transition tree. Then each belief state in $\widehat{\Phi}(T, ext(\Delta))$ is an extension of a DNF-state in $\widehat{\Phi_{DNF}}(T, \Delta)$, and each DNF-state in $\widehat{\Phi_{DNF}}(T, \Delta)$ has an extension in $\widehat{\Phi}(T, ext(\Delta))$.*

[1] A complete version of this paper with proofs and detailed algorithms along with the system can be found at `www.cs.nmsu.edu/~sto`.

This theorem together with Propositions 2 and 3 allows DNF$_{ct}$ to compute solutions of contingent planning problems efficiently by using a compact DNF form to represent belief states. The next section describes a novel, efficient AND/OR forward search algorithm, called PrAO, implemented in DNF$_{ct}$ for finding contingent solutions.

PrAO Algorithm for Contingent Planning

In this section, we describe an algorithm, called PrAO, for contingent planning. PrAO is similar to a standard AND/OR graph search algorithm in the way it generates and maintains an AND/OR graph during the search. The key difference between PrAO and others lies in the way PrAO keeps track of potential solutions and eliminates useless nodes of the graph.

We start by introducing the notion of a search graph, that is similar to the notion of AND/OR graph in the literature (Martelli and Montanari 1973) and more convenience for use with the DNF-representation.

Definition 3. *Given a contingent planning problem* $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$*, a* search graph *is a labeled transition graph* $(\mathcal{S}, \mathcal{T}, s_0)$ *where*

- $\mathcal{S}$ *is a set of DNF-states, each DNF-state is referred to as a node;*
- s_0 *is the initial DNF-state representing* I*,* $s_0 \in \mathcal{S}$*; and*
- $\mathcal{T}$ *is a set of transitions,* $\mathcal{T} \subseteq \mathcal{S} \times (A \cup \Omega) \times \mathcal{S}$*, such that*
  - *for every* $\omega \in \Omega$*, if* $(s, \omega, s_1) \in \mathcal{T}$ *then* $s \models pre(\omega)$*, and there exists a unique,* dual *to* (s, ω, s_1)*,* $(s, \omega, s_2) \in \mathcal{T}$ *such that* $\{s_1, s_2\} = \{s^+_\omega, s^-_\omega\}$*; and*
  - *for every* $a \in A$*, if* $(s, a, s_1) \in \mathcal{T}$ *then* $s \models pre(a)$ *and* $s_1 = \Phi_{DNF}(a, s)$*.*

Intuitively, a search graph represents the transition graph that we have explored so far during the search. Each node encodes a DNF-state, the initial DNF-state s_0 is the root node, and $\mathcal{T}$ is the set of transitions between nodes (belief states). Observe that the search graph is in general not a tree as a node may have multiple incoming edges.

A transition (s, t, s') is called an *or-edge* (resp. *and-edge*) if $t \in A$ (resp. $t \in \Omega$). For convenience, we often refer to a transition $(s, t, s') \in T$ as an *outgoing edge* from s (or *incoming edge* to s'). For a DNF-state Δ and an action $a \in A$, we refer to $\Phi_{DNF}(a, \Delta)$ as the successor state of Δ after the execution of a; similarly, for $\omega \in \Omega$, Δ^+_ω and Δ^-_ω are called successor states of Δ after the execution of ω.

The description of PrAO needs the following notations:

Definition 4. *A node* s *is* goal-reachable*, or* goal *for short, if (*i*)* $s \models G$*; or (*ii*) there is an or-edge* (s, a, s') *in* $\mathcal{T}$ *and* s' *is goal-reachable; or (*iii*) there exist dual and-edges* (s, ω, s_1) *and* (s, ω, s_2) *in* $\mathcal{T}$ *and both* s_1 *and* s_2 *are goal-reachable.*

During the search, a node s in $\mathcal{S}$ can be in one of the following states. When s is a new node added to $\mathcal{S}$, it is marked as *unexplored*. If s is selected for expansion, it becomes an *explored* node. Furthermore, s can be a goal node (Definition 4) or a *dead node*, defined as follows:

Definition 5. *An explored node* s *is a* dead node *if*

- $s \not\models G$ *and there is no outgoing edges from* s*; or*
- *for every or-edge* (s, a, s') *in* $\mathcal{T}$*,* s' *is dead; and for every pair of and-edges* (s, ω, s_1) *and* (s, ω, s_2) *in* $\mathcal{T}$ *at least one of the two nodes* s_1 *and* s_2 *is dead.*

Intuitively, if the root s_0 becomes a goal node then a solution is detected. If s_0 becomes dead then the problem has no solution. When a node is being expanded, the state of the nodes in $\mathcal{S}$ is updated by some propagation procedures, as presented later, based on Definitions 4-5 and a pruning technique, which is built based on the following observations:

- If a node s becomes goal via an or-edge (s, a, s') or via two dual and-edges (s, ω, s_1) and (s, ω, s_2), then all the other outgoing edges from s can be removed from $\mathcal{T}$.
- If s is dead, then every (s', t, s) in $\mathcal{T}$, and its dual $(s', t, s")$ if $t \in \Omega$, become(s) useless and can be removed.
- When a transition (s', t, s) is removed from $\mathcal{T}$, s and the nodes that can be reached from s may no longer be reached from s' and thereby from the root s_0. For expansion, hence, we consider only the unexplored nodes that are *active*, defined as follows:

Definition 6. *A node* s *in* $\mathcal{S}$ *is said to be* active *if*

- s *is the root node; or*
- *there exists an edge* (s', t, s) *in* $\mathcal{T}$ *and* s' *is active.*

A node that is not active is said to be disabled*.*

Clearly, only active nodes can be reached from s_0 and dead nodes are not active due to the second observation.

Given a contingent problem $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$, PrAO starts with the graph $(\{s_0\}, \emptyset, s_0)$, where s_0 is the root node representing I, and iteratively constructs a search graph $(\mathcal{S}, \mathcal{T}, s_0)$ until a solution has been found or it is determined that $\mathcal{P}$ has no solution. Initially, s_0 is marked as unexplored.

At each iteration, PrAO executes the following tasks:

- Select an active unexplored node s with the best heuristic value in $\mathcal{S}$ for expansion. If no such node exists then terminate the search with no solution;
- Expand node s and update the graph accordingly; and
- Check the state of the root node s_0: if s_0 becomes a goal node then extract and return the solution. If s_0 is a dead node then terminate the search with no solution.

The expansion phase is as follows:

- For each action a in A such that $s \models pre(a)$: compute $s' = \Phi_{DNF}(a, s)$. If s' already exists in $\mathcal{S}$ and s' is dead then ignore this transition. Otherwise, if s' is not in $\mathcal{S}$ then: extend $\mathcal{S}$ with s' and mark s' as active, unexplored (resp. goal) if $s' \not\models G$ (resp. $s' \models G$). Add (s, a, s') to $\mathcal{T}$. If s' is a goal node (either new or existing) then: mark s as a goal node, execute $goal\_propagation(s, a)$, and return. Otherwise, execute $reactivation\_propagation(s')$.
- For each observation ω in Ω such that $s \models pre(\omega)$, $s \not\models \ell(\omega)$, and $s \not\models \overline{\ell(\omega)}$: compute $s_1 = s^+_\omega$ and $s_2 = s^-_\omega$. If either s_1 or s_2 exists in $\mathcal{S}$ and it is a dead node then ignore this transition. Otherwise, for $i = 1, 2$, if s_i is not in S then mark s_i as active, unexplored (resp. goal) if $s_i \not\models G$ (resp. $s_i \models G$). Extend $\mathcal{T}$ with (s, ω, s_i).
  If both s_1 and s_2 are goal then: mark s as a goal node and execute $goal\_propagation(s, \omega)$, and return. Otherwise, execute $reactivation\_propagation(s_i)$, for $i = 1, 2$.
- After the first two steps, if there does not exist such (s, t, s') in $\mathcal{T}$ then mark s as a dead node and execute $dead\_propagation(s)$. Otherwise, mark s as explored.

The procedures used in the expansion phase are as follows:

- $goal\_propagation(s, t_0)$: For every edge (s, t, s') in $\mathcal{T}$ such that $t \neq t_0$, remove (s, t, s') from $\mathcal{T}$ and exe-

cute $isolation\_propagation(s')$. For each edge (s_1, t, s) in $\mathcal{T}$: if $t \in A$ then mark s_1 as goal and execute $goal\_propagation(s_1, t)$. Otherwise ($t \in \Omega$), let (s_1, t, s_2) be the other and-edge in $\mathcal{T}$ ($s_2 \neq s$), if s_2 is also goal then mark s_1 as goal and execute $goal\_propagation(s_1, t)$.

- ◇ $dead\_propagation(s)$: For every edge (s_1, t, s) in $\mathcal{T}$: (*i*) remove (s_1, t, s) from $\mathcal{T}$; (*ii*) if $t \in \Omega$ then remove also its dual and-edge (s_1, t, s_2) from $\mathcal{T}$ ($s_2 \neq s$) and execute $isolation\_propagation(s_2)$. If there is no more outgoing edges from s_1 in $\mathcal{T}$ then mark s_1 as a dead node and execute $dead\_propagation(s_1)$.
- ◇ $isolation\_propagation(s)$: If s is a goal node then return. $\mathcal{T}$ does not contain an incoming edge to s from an active node then (*i*) mark s as disabled, and (*ii*) for each $(s, t, s")$ in $\mathcal{T}$ execute $isolation\_propagation(s")$.
- ◇ $reactivation\_propagation(s)$: If s is active then return. Otherwise, mark s as active and, for each (s, t, s') in $\mathcal{T}$, execute $reactivation\_propagation(s')$.

Observe that $isolation\_propagation(s)$ temporarily disables the subgraph each node on which is disconnected from the main graph by $goal\_propagation$ or $dead\_propagation$, and prevents the expansion of the unexplored nodes on this subgraph. In contrast, $reactivation\_propagation(s)$ enables the node s and every node that can be reached from s, if they are disabled, whenever a new edge connects them to the main graph again. This makes the pruning technique safe and thereby PrAO complete.

Proposition 4. *For each goal node s in the search graph $(\mathcal{S}, \mathcal{T}, s_0)$, one of the following conditions holds:*

- *there exists no outgoing edge, if $s \models G$; or*
- *there exists a unique edge $(s, a, \Phi_{DNF}(a, s))$ in $\mathcal{T}$, where $a \in A$ and $\Phi_{DNF}(a, s)$ is a goal node; or*
- *there exists a pair of edges (s, ω, s^+_ω) and (s, ω, s^-_ω) in $\mathcal{T}$, where $\omega \in \Omega$ and both s^+_ω and s^-_ω are goal nodes.*

Thanks to Proposition 4, if we ignore the incoming edges to the active goal nodes from disabled nodes, then the remaining graph becomes the solution tree of the contingent problem $\mathcal{P}$. Formally, given a contingent planning problem $\mathcal{P}$ and a search graph $(\mathcal{S}, \mathcal{T}, s_0)$ constructed by PrAO and a goal node $s \in \mathcal{S}$, by $tree(s)$ we denote the following transition tree:

- $tree(s) = []$ if $s \models G$;
- $tree(s) = a \circ tree(s_1)$ if $(s, a, s_1) \in \mathcal{T}$ and $a \in A$; and
- $tree(s) = \omega(tree(s_1)|tree(s_2))$ if (s, ω, s_1) and (s, ω, s_2) belong to $\mathcal{T}$ and $\omega \in \Omega$.

We can prove the following property of PrAO:

Proposition 5. *Given a contingent planning problem $\mathcal{P}$ and a search graph $(\mathcal{S}, \mathcal{T}, s_0)$ constructed by PrAO, it holds that*

- *If s_0 is a goal node then $tree(s_0)$ is a solution of $\mathcal{P}$.*
- *If s_0 is a dead-end node, or s_0 is not a goal node and there exists no unexplored node in $\mathcal{S}$, then $\mathcal{P}$ does not have a solution.*

To illustrate the algorithm, let us consider the problem $\mathcal{P} = \langle \{f, g, h\}, \{a, b, c, d, e, t, p_1, p_2\}, \{s_g\}, true, \{f, \neg g, h\} \rangle$. The specification of the actions are given by

- $pre(a) = \emptyset$, $O(a) = \{\emptyset \rightarrow f\}$; [2]
- $pre(b) = \{f\}$, $O(b) = \{\emptyset \rightarrow \{\neg f, g\}\}$;

[2] We simplify the writing as $O(a)$ contains only one outcome.

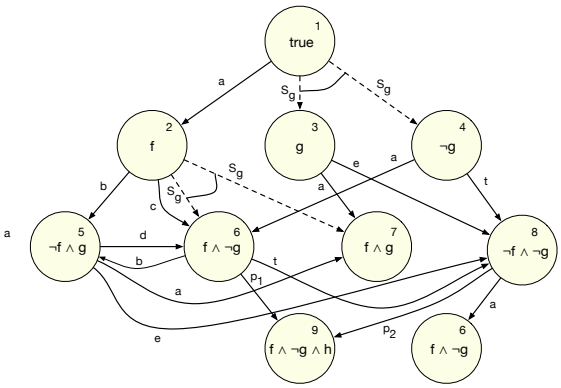

Figure 2: Search Graph for $\mathcal{P}$

- $pre(c) = \{f\}$, $O(c) = \{\emptyset \rightarrow \{f, \neg g\}\}$;
- $pre(d) = \{\neg f, g\}$, $O(d) = \{\emptyset \rightarrow \{f, \neg g\}\}$;
- $pre(e) = \{g\}$, $O(e) = \{\emptyset \rightarrow \{\neg f, \neg g\}\}$;
- $pre(t) = \{\neg g\}$, $O(t) = \{\emptyset \rightarrow \{\neg f, \neg g\}\}$;
- $pre(p_1) = \{f, \neg g\}$, $O(p_1) = \{\emptyset \rightarrow h\}$;
- $pre(p_2) = \{\neg f, \neg g\}$, $O(p_2) = \{\emptyset \rightarrow \{f, h\}\}$;
- $pre(s_g) = \emptyset$ and $l(s_g) = g$.

It is easy to see that the problem has different solutions. Among them are the following transition trees: (*i*) $a \circ b \circ d \circ p_1$; (*ii*) $a \circ c \circ p_1$; or (*iii*) $s_g(e \circ p_2 \mid t \circ p_2)$.

The search graph constructed by PrAO for obtaining the above transition trees is given in Figure 2. Depending on the order of nodes selected during its execution, PrAO would return different solutions. For illustrative purpose, let us look at this in more details. For simplicity, we will refer to a node by its number. In the figure, solid links represent or-edges and dash links represent and-edges.

Initially, node 1 is created. The expansion of 1 leads to the creation of nodes 2, 3, and 4 (and their addition to $\mathcal{S}$). Node 1 becomes explored. None of the new nodes satisfies the goal, so goal-propagation is not triggered. Node 1 has some outgoing edges, so no dead-end propagation is called. Hence, no isolation propagation is executed either. Since no successor node of 1 exists in $\mathcal{S}$, no reactivation-propagation is called. We consider two orders of expansion by PrAO:

- Assume that we expand nodes 2, 5, 6 in this order, and each expansion is executed in the order the actions are given. Expanding 2 results in the creation of 5, 6, 7 with the corresponding edges. No propagation is executed.
  Expanding 5 creates $(5, a, 7)$, $(5, d, 6)$, and (8, $(5, e, 8)$). Reactivation-propagations at 6 and 7 do not change anything since they are not disabled.
  Expanding 6 creates $(6, b, 5)$, $(6, t, 8)$, and (9, $(6, p_1, 9)$). Since 9 satisfies the goal, it is marked as goal and so is 6 as p_1 is an or-edge. Then $goal\_propagation(6, p_1)$ is executed, which removes $(6, b, 5)$, $(6, t, 8)$. 5 and 8 still have another incoming edge so they are not disabled by isolation-propagation at this point. If 5 is considered before 2 then 5 becomes goal first, the edge $(5, d, 6)$ is retained and the edges $(5, a, 7)$ and $(5, e, 8)$ are removed. 8 is disabled by the isolation-propagation as it does not have an incoming edge. Then $goal\_propagation(2, b)$ removes every outgoing edges from 2 but $(2, b, 5)$. 7 now

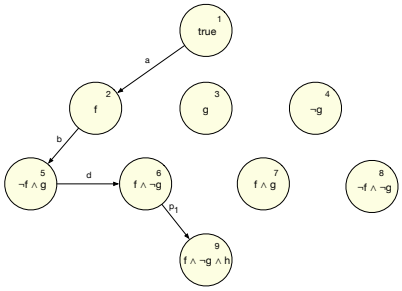

Figure 3: First solution: $a \circ b \circ d \circ p_1$

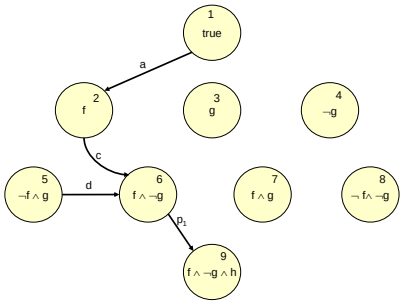

Figure 4: Second solution: $a \circ c \circ p_1$

is disabled. Then 1 becomes goal and 3 and 4 are disabled with no incoming edges. The first solution is found with the remaining graph illustrated in Figure 3.
Similarly, if 2 is considered before 5 (inside $goal\_propagation(6)$), then the second solution is returned with the remaining graph as in Figure 4.

- Suppose that the order of expansion is 3 and 8. The expansion of 3 creates 7 and 8. Expanding 8 creates 6 and 9 and edges from 8 to them. 9 and 8 become goals. Goal-propagation removes $(8, a, 6)$, disables 6, marks 3 as goal, removes $(3, a, 7)$, and disables 7. Suppose that 4 is better than 2 and worse than 6 and 7, in terms of heuristic. PrAO selects 4 to expand next, since 6 and 7 are disabled. Expanding 4 creates two edges from 4 to 6 and 8, activating 6 and making 4 a goal node. The goal propagation results in the graph 5 and the third solution is achieved.

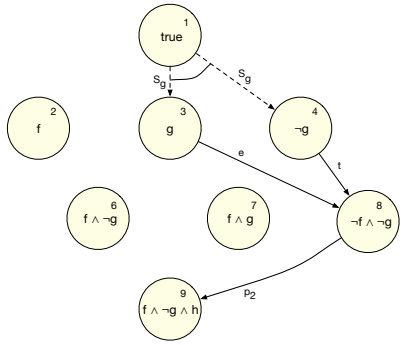

Figure 5: Third solution: $s_g(e \circ p_2 \mid t \circ p_2)$

DNF$_{ct}$ Planner and Empirical Evaluation

Implementation of DNF$_{ct}$: DNF$_{ct}$ implements PrAO on top of DNF (To et al. 2009). The heuristic function used in DNF$_{ct}$ for the experiments is same as that used in DNF, i.e., a combination of the number of satisfied subgoals and the number of partial states in the DNF-state. We did try with the heuristic function similar to that described in (Hoffmann and Brafman 2005) but the results did not improve.

Planners: We compare DNF$_{ct}$ with CLG (Albore et al. 2009), contingent-FF (Hoffmann and Brafman 2005), and POND 2.2 (Bryce *et al.* 2006) obtained from their websites. These planners are known to be among the best currently available contingent planners. We executed contingent-FF with both options, i.e., with and without helpful actions, and report here the best result for each instance. POND was executed using the AO* search algorithm (aostar). We observe 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 of several execution times for each instance.

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

Benchmarks: We tested the planners with (*i*) different variants of the following domains: bomb in the toilet (e.g., bts, $btnd$), $block$, $logistic$, $grid$, and $unix$; (*ii*) variants of the grid domains such as $cballs$-n-m, $doors$-n, $localize$-n, and $wumpus$-n. The first group of domains is from the distributions of contingent-FF and POND, while the second comes from the CLG distribution. We add to the second group some of our own by modifying some challenging conformant domains to force planners to generate conditional plans. Those domains are $edispose$, $e1d$, and $epush$, modified from $dispose$, 1-$dispose$, and $push$ respectively. The modifications are as follows. For example, in the $dispose$ domain, the action $pickup(o, p)$ is given by $(\{at(p)\}, \{obj\_at(o, p) \rightarrow holding(o) \wedge \neg obj\_at(o, p)\})$; in its variant $edispose$, the sensing action $sense(o, p)$ with $\ell(sense(o, p)) = obj\_at(o, p)$ and $pickup(o, p)$ is changed to $(\{at(p), obj\_at(o, p)\}, \{true \rightarrow holding(o) \wedge \neg obj\_at(o, p)\})$, meaning that $pickup(o, p)$ is applicable only if both the agent and the object o are at location p.

Summary of Experimental Results: Tables 1-2 report the experimental results in the form of $t(s/d)$, where t, s, and d denote the overall execution time, the number of actions in the solution, and the depth of the solution tree respectively. Since POND does not report the depth of a solution (d), we omit that in the results of POND. Usually, d and s are criteria for evaluation of the quality of a solution[3]. In these tables, OM denotes out-of-memory, TO means time-out, E indicates incorrect report, MC stands for "too many clauses" of large instances for contingent-FF to handle, NA refers to the instances that are non-applicable for the planner (e.g., NA is in CLG's column on $btnd$ because CLG does not consider non-deterministic actions).

We observe that even though DNF$_{ct}$ employs a best first search in AND/OR graph for solutions, not an AO* search

[3]We consider d to be more important, as it is the maximum number of actions the agent needs to execute to obtain the goal.

as contingent-FF and POND use[4], the quality of the solutions found by DNF$_{ct}$ is not worse than those found by the other planners, in general. We suspect that this is because contingent-FF and POND use an inadmissible heuristic function. Moreover, the heuristic based on the number of actions in a solution of a relaxed problem may be very inaccurate in the presence of incomplete information.

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

Problem	DNFct	CLG	contingent-FF	Pond
block-3	0.58 (5/4)	0.08 (6/4)	0.02 (6/4)	**0.01 (5/4)**
block-7	11.6 (69/28)	4.58 (55/9)	**0.46** (49/12)	OM
block-11	OM	**35.68 (115/18)**	TO	OM
btcs-70	**2.77 (139/70)**	13.7 (140/140)	123.64 (139/70)	74.04 (139)
btcs-90	**5.4 (179/90)**	40.71 (180/180)	476.8 (179/90)	TO
btcs-150	**27.5 (299/150)**	379 (300/300)	MC	TO
btnd-70	**1.71** (276/72)	NA	536.6 (140/72)	TO
btnd-90	**2.78** (356/92)	NA	2070 (180/92)	TO
btnd-150	**7.47 (596/152)**	NA	TO	TO
bts-70	**2.25** (139/70)	6.43 (70/70)	1672 (70/70)	TO
bts-90	**3.21** (179/90)	18.96 (90/90)	TO	TO
bts-150	**8.15** (299/150)	200 (150/150)	TO	TO
ebtcs-70	**1.21 (139/70)**	24.79 (209/71)	63 (139/70)	24.69 (139)
ebtcs-90	**1.57 (179/90)**	69.99 (269/91)	255.5 (179/90)	TO
ebtcs-150	**4.19 (299/150)**	603.3 (449/150)	MC	TO
ebtnd-70	**1.49** (276/72)	NA	16.3 (208/72)	TO
ebtnd-90	**3.19** (356/92)	NA	53.15 (268/92)	TO
ebtnd-150	**6.25 (596/152)**	NA	MC	TO
elogistics-5	0.92 (301/179)	0.08 (147/21)	**0.02** (156/23)	0.67 (143)
elogistics-7	1.31 (422/126)	0.11 (210/22)	**0.04** (223/23)	0.95 (212)
elogistics-L	OM	**90.3 (36152/73)**	TO	OM
grid-3	1.97 (313/51)	0.94 (114/30)	**0.06 (23/23)**	104 (178)
grid-4	2.9 (982/71)	4.64 (872/51)	**0.14 (49/49)**	OM
grid-5	12.59 (1337/81)	1.87 (212/40)	**0.15** (46/46)	OM
medpks-70	**1.49** (141/72)	7.51 (141/71)	968.6 (140/71)	TO
medpks-90	**2.69 (199/101)**	24.35 (199/101)	TO	TO
medpks-150	**6.49 (299/151)**	103.94 (299/151)	TO	TO
unix-2	0.78 (48/37)	0.64 (50/39)	**0.13 (48/37)**	1.71 (48)
unix-3	**2.02 (111/84)**	5.88 (113/86)	3.84 (111/84)	OM
unix-4	**16.26 (238/179)**	84.42 (240/181)	142.8 (238/179)	OM

Table 1: Problems from contingent-FF and POND distributions.

Domains in contingent-FF **and** POND **Distributions**: Table 1 reports the results of our experiments from the domains in contingent-FF and POND distributions. As one can see, DNF$_{ct}$ outperforms all the other planners on seven out of ten domains. DNF$_{ct}$ also scales up very well on these domains as it can solve the largest instances within a small total runtime, while the other planners cannot solve or spent much longer time for a solution for those instances. The solution trees for these problems found by DNF$_{ct}$ are comparable to those found by the other planners. Observe that DNF$_{ct}$ lost to the other planners on the smallest instance of $unix$, a domain it is strong at. This is due to the overhead of the translation process of the input theory incurred in DNF$_{ct}$. For most small instances, DNF$_{ct}$ spent most time on the translation process, e.g., for $unix$-2 DNF$_{ct}$ spent only 0.025 second for the search while 0.75 second for the translation. In this group, DNF$_{ct}$ does not perform well on $block$, $elogistic$, and $grid$. One of the reasons is that the heuristic function based on the number of satisfied subgoals, that DNF$_{ct}$ uses, is misleading on these problems.

[4]POND supports several options for search algorithms, we selected AO* search option in our experiments.

Problem	DNFct	CLG	cont-FF	Pond
cball-3-2	**0.99** (609/40)	2.91 (2641/34)	TO	2.2 (597)
cball-3-3	**10.5** (8042/57)	62.5 (60924/48)	TO	39.7 (4808)
cball-5-1	**1.1** (117/68)	1.3 (586/65)	TO	527 (199)
cball-5-2	**21.5** (5132/113)	167 (72817/107)	TO	OM
cball-8-1	**10.42 (279/148)**	43.1 (2411/171)	TO	OM
cball-8-2	**768.7 (27472/292)**	TO	TO	OM
edisp-3-2	**0.64 (397/36)**	0.46 (752/39)	E	TO
edisp-3-3	**1.58 (3891/57)**	5.77 (8552/52)	E	TO
edisp-5-1	**0.99** (98/60)	3.1 (177/60)	E	TO
edisp-5-2	**3.45** (2753/97)	27.4 (7993/87)	E	TO
edisp-10-1	**46.9 (400/233)**	140 (1051/237)	E	TO
edisp-10-2	**298.3 (31357/402)**	TO	E	TO
e1d-3-1	**1.15 (33/20)**	2.71 (50/20)	E	TO
e1d-3-3	**1.74** (3557/88)	202 (15294/62)	TO	TO
e1d-5-1	**1.57** (99/59)	52.25 (200/58)	TO	TO
e1d-5-3	**88.5 (76088/417)**	TO	TO	TO
e1d-10-1	**164 (399/233)**	TO	TO	TO
e1d-10-2	**461 (47652/665)**	TO	TO	TO
doors-7	**4.28** (2193/53)	7.6 (2153/51)	E	17.99 (2159)
doors-9	**57.6 (44998/89)**	585 (46024/95)	E	1262 (44082)
localize-5	**0.57 (49/23)**	1.86 (112/24)	42 (53/53)	TO
localize-7	**0.72 (80/36)**	6.89 (231/37)	MC	TO
localize-9	**0.84** (113/53)	21.2 (386/50)	MC	TO
localize-11	**1.2 (144/62)**	63.72 (577/63)	MC	TO
localize-13	**1.56 (176/75)**	E	MC	TO
epush-3-1	0.53 (39/29)	**0.39 (50/24)**	0.6 (61/37)	TO
epush-3-3	**1.32** (1249/87)	7.04 (6196/44)	TO	TO
epush-6-2	**9.76** (5001/241)	447 (24523/148)	TO	TO
epush-6-3	**88 (103065/383)**	TO	TO	TO
epush-10-1	**55.2 (731/268)**	341 (1983/446)	TO	TO
epush-10-2	**392 (64808/845)**	TO	TO	TO
wumpus-5	2.47 (1083/34)	**1.13** (754/41)	E	4.65 (587)
wumpus-7	55.15 (29853/74)	**9.57 (6552/57)**	E	TO
wumpus-10	OM	**2954 (280k/100)**	E	TO

Table 2: Challenging problems from CLG distribution and modification of conformant problems

POND is the best at $block$-3 and able to solve several small instances, but its overall performance is poor. Contingent-FF outperforms the other planners on all instances of $grid$ domain and several small instances of some other domains; including $block$-7, $elogistic$-5, and $elogistic$-7, and $unix$-2. This planner is also the second best, behind DNF$_{ct}$, on $btnd$ and $ebtnd$. Overall, CLG is the second best planner as it is the only one that can solve the largest instances of $block$ and $elogistics$, and performs second best on most of other domains, except $btnd$ and $ebtnd$.

Challenging Problems: Table 2 contains the results of our experiments with the set of challenging problems, which are proposed by the authors of CLG and by us. As it can be seen, these problems are much harder than those reported in Table 1. contingent-FF and POND can only solve few small

instances and POND is a little better than contingent-FF in this group of problems. Again, CLG is the only competitor of DNF$_{ct}$ on these problems. Out of all four domains proposed by the authors of CLG, DNF$_{ct}$ outperforms all the other planners on three ($cball$-n-m, $doors$-n, and $localize$-n). On the other hand, CLG is best in $wumpus$-n. DNF$_{ct}$ also has the best performance on the modifications of challenging conformant problems: $edis$-n-m, $e1d$-n-m, and $epush$-n-m. Observe that, CLG scales up well on the first dimension (n) of $cball$-n-m, $edis$-n-m, and $epush$-n-m but it has trouble with the second dimension, for which DNF$_{ct}$ does much better. Note that on these domains, n and m denote the number of locations and the number of objects, respectively. Initially, the location of each object o_i ($i = 1, \ldots, m$) is uncertain among n given places and described by the literal $at(o_i, p_j)$ ($j = 1, \ldots, n$). Thus the size of the problem is at least linear in $n \times m$ and the number of states in the initial belief state is linear in n^m, i.e., exponential in m. This explains why the other planners, including CLG, scale poorly on m. This also confirms the superior scalability of DNF$_{ct}$ compared to the others, in general. We believe that DNF$_{ct}$ underperforms CLG on $wumpus$-n because of two reasons. First, the goal of each instance of this domain contains only one subgoal that needs to be obtained (the other subgoal already exists in the initial belief state and never disappears). Thus, the heuristic function based mostly on the number of subgoals used in DNF$_{ct}$ does not help much for the search to find a solution. Second, each problem instance of this domain contains a large number of disjunctive clauses (or-clauses), making the size of the initial DNF-state very large. For example, the initial DNF-state of $wumpus$-10 contains 2,567,504 partial states.

Effectiveness of the Pruning Technique: We would like to conclude this section with a discussion of the effectiveness of the pruning technique (implemented by the isolation/reactivation propagation procedures) incorporated in PrAO. Table 3 reports the results of the performance of DNF$_{ct}$ on several problems with/without using the pruning technique. On most problems, using the pruning technique results in a better performance of DNF$_{ct}$, i.e., it spends less time searching for a solution (the translation time is the same so it is not included) as it generates and expands less nodes. In addition, the solution tree is better when the pruning technique is used, in general. We also observe that there are few problem instances on which application of the pruning technique makes the performance of DNF$_{ct}$ worse, e.g., $wumpus$-7. Nevertheless, this technique contributes significantly to the good overall performance of DNF$_{ct}$, besides the DNF-representation.

problem	search_t	s/d	gen./exp.	search_t	s/d	gen./exp.
block-7	10.7	69/28	42.7k/42.1k	11.75	948/733	55.4k/45.6k
btnd-150	5.88	596/152	12.8k/713	80.48	893/153	24.5k/893
cball-8-2	735.7	27.4k/292	123k/61.9k	1683	15.7k/272	264.7k/264k
doors-9	34	445k/89	131.7k/93.5k	67.4	45k/89	248.6k/242k
push-6-2	6.83	4991/241	25.4k/14.4k	77.94	7625/338	265k/263.8k
wumpus-7	47.27	29.8k/74	134k/96.3k	28.5	26.8k/109	61k/34k

Table 3: Results of DNF$_{ct}$: with pruning (left) v.s. without pruning (right). search_t: search time, s/d: size/depth of the solution, gen./exp.: number of generated/expanded nodes

Conclusion

This paper presented a new approach to contingent planning using a DNF representation of belief states, along with PrAO, a novel variant of AND/OR search algorithms which includes techniques for pruning the search space. While the DNF-representation provides a compact encoding of belief states and fast state computation, PrAO's goal is to minimize the number of nodes generated during the search. We developed a new planner DNF$_{ct}$ using these techniques and experimentally evaluated it against state-of-the-art contingent planners. The results showed that DNF$_{ct}$ is very competitive with other planners and scales up better in many domains.

Although DNF$_{ct}$ exhibits good performance and scalability, several open questions remain. For example, we observe that there are a few problems on which disjunctive representation appears to be unsuitable (e.g., $wumpus$-n), or the pruning technique has negative effectiveness, e.g., $wumpus$-7. As such, for problems rich in disjunctive information, another representation might be needed (e.g., conjunctive normal form). Furthermore, the effectiveness of the pruning technique may need to be investigated further.

References

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

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

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

R. Brafman, J. Hoffmann. Conformant planning via heuristic forward search: A new approach. *ICAPS*, 2004.

R. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Comp. Surv.*, 24(3), 1992.

D. Bryce et al. Planning Graph Heuristics for Belief Space Search. *JAIR*, 26:35–99, 2006.

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

A. Martelli and U. Montanari. Additive and/or graphs. In *IJCAI*, 1973.

M. Peot and D.E. Smith. Conditional nonlinear planning. In *Proc. of 1st Int. Conf. on AIPS*, 1992.

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

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

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

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