A New Approach to Contingent Planning Using A Disjunctive Representation in AND/OR forward Search with Novel Pruning Techniques

Son Thanh To

Department of Computer Science
New Mexico State University
sto@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 proposed by To, Pontelli, and Son for conformant planning, for the search for solutions in the belief space. In the previous work for conformant planning, a complete transition function for computing successor belief states under the representation 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 for computing successor belief states efficiently, i.e., in polynomial time under reasonable assumptions. The paper also presents a novel variant of a standard AND/OR search algorithm which allows the planner to significantly prune the search space; furthermore, by the time a solution is found, the remaining search graph is also a solution tree for the contingent planing problem, thanks to the pruning. The strength of these techniques is confirmed by the empirical results obtained from a large set of most benchmarks available in the literature.

Keywords: Planning, Contingent Planning, Belief State Representation, Transition Function, Disjunctive Normal Form Formula, AND/OR Search, Pruning

1. Introduction

Contingent planning [22, 15, 16] 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 [9, 2, 17].

One of the best-known and efficient approaches to contingent planning is to transform the problem into a heuristic AND/OR search problem in the belief state space [4]. 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, the use of *binary decision diagrams* (BDDs) [6] was proposed to represent belief states in a model checking based planner, called MBP [3]. Later, Bryce, Kambhampati, and Smith 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[7]. 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 the BDD representation is still very large and sensitive to the order of the variables. Moreover, computing successor belief states in the BDD 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 several previous works [10, 1, 21] and in this work (with POND).

At the other extreme of the use of belief states (that explicitly enumerates all possible states of the world), Hoffmann and Brafman represent 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 [5, 10, 11]. This approach does not store the knowledge about the state of the world in memory, except the known propositions and the corresponding action sequence. The advantage of this implicit representation is that 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 computations, as it needs to reason about CNF formulae that capture the semantic of the entire action sequences to the belief states from and in conjunction with the initial belief state. Furthermore, checking whether a proposition holds after the execution of even one single action in the presence of incomplete information is co-NP complete. This is one of the main 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 conditions of the conditional effects as observed from the experimental results of several works [1, 19, 21].

A translated-based approach to contingent planning were proposed by Albore, Palacios, and Geffner [1]. This approach was extended from that of Palacios and Geffner, proposed in conformant planning [13, 14], that translates a conformant planning problem into a classical planning problem and uses the well-known off-the-shelf classical planner FF [12] as the underlying planner to solve the problem. The extended approach [1] translates a contingent problem into a non-deterministic search problem in the state space whose literals represent the beliefs over the original problem. The non-deterministic problem then is relaxed to be a classical planning problem. This approach assumes that the uncertainty lies in the initial world only and all actions are deterministic. The work showed that the translation is polynomial under some conditions. This approach demonstrates a great improvement, i.e., the resulting planner CLG can solve hard problems of large size, compared to the previous approaches. However, the translation and the number of literals in the resulting 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. Moreover, this approach does not consider the problems with any disjunctions in the goal.

Our previous work in conformant planning brought a different perspective to deal with incomplete information by using a special, compact form of disjunctive formulae, called *minimal DNF*, to represent belief states. The work defined a direct complete transition function for computing the successor belief states encoded in this representation in the presence of incomplete information efficiently, i.e., polynomial under reasonable assumptions [18]. The advantage of this method is confirmed by the fact that our resulting conformant 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 a compact conjunctive normal form (CNF) formula, called minimal-CNF, and prime implicates as other representations of belief states [19, 20].

Recently, we proposed a new approach to contingent planning by the use of the minimal-DNF representation of belief states in a new AND/OR forward search algorithm for contingent solutions [21]. The work extended the transition function, defined in the prior work [18] for conformant planning, to also han-

dle uncertain action effects and sensing actions required in contingent planning. On the other hand, a novel variant of AND/OR forward search algorithms, called PrAO, was developed for contingent planning. Key to PrAO is a novel safe pruning technique which can significantly reduce the search space for most problems. Furthermore, the solution extraction in PrAO is fairly simple, as the remaining search graph, by the time a solution is found, is also a solution tree, thanks to the pruning technique. The PrAO algorithm and the extended minimal-DNF representation were implemented in a contingent planning system, called DNF$_{ct}$. The experiments show that DNF$_{ct}$ offers a very competitive performance compared with other state-of-the-art contingent planners on a large set of benchmarks.[1]

This work completes and extends the aforementioned work [21] in different ways. It provides the proofs for most results relevant to the minimal-DNF representation, that were not presented in the previous works [18, 21]. For completeness, the paper presents underpinnings of contingent planning as AND/OR forward search, a standard AND/OR forward search algorithm for contingent planning solutions and proofs for the correctness of the algorithm, as the foundation for PrAO. Then the paper presents the PrAO algorithm, extended from the standard algorithm by incorporating the minimal-DNF representation and the pruning techniques. The proofs for soundness and completeness of PrAO are also provided. For the study of the effectiveness of the pruning techniques, PrAO is experimentally compared with a variant of PrAO, where the pruning techniques are not applied, on most of the benchmarks. The experiments show that the pruning eliminates a large potion of the search space and improves the performance significantly for most problems. Like several recent works in contingent planning [10, 1], this work does not consider contingent solutions with loops. An AND/OR forward search algorithm that allows solutions with loops, e.g., LAO* [8], can be used to solve Markov decision problems.

The paper is organized as follows. Next section presents the background of contingent planning. Section 3 reviews the minimal-DNF representation and extends it for handling non-deterministic and sensing actions. Section 4 presents a standard AND/OR forward search algorithm for contingent planning solutions. Section 5 presents PrAO, extending the standard algorithm by incorporating the minimal-DNF representation and the pruning techniques. Section 6 reports and discusses the experimental results. Section 7 summarizes the main results of the paper.

2. Background: Contingent Planning

2.1. 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, $\bar{L} = \{\bar{\ell} \mid \ell \in L\}$. 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 in $O(a)$ is a set of conditional effect of the form $\psi \rightarrow \ell$ (also written as $o : \psi \rightarrow \ell$), where ψ is a set of literals, called an *e-condition* of o, and ℓ is a

[1]The systems and the benchmarks used for the experiments in this paper can be downloaded from `http://www.cs.nmsu.edu/~sto`.

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 $s \in S$. A ℓ is said to be *known* in S if either $S \models \ell$ or $S \models \overline{\ell}$ holds.

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 (o occurs during the execution of a) is

$$e(o, s) = \{\ell \mid \psi \rightarrow \ell \in o, s \models \psi\} \tag{1}$$

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

$$\Phi(a, S) = \begin{cases} \bigcup_{o \in O(a)} \{s \setminus \overline{e(o, s)} \cup e(o, s) \mid s \in S\} & \text{if } S \neq \emptyset \wedge S \models pre(a) \\ undefined & \text{otherwise} \end{cases} \tag{2}$$

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 and S be a belief state. The application of ω in S results in a pair of two belief states, denoted by S^+_ω and S^-_ω and defined as

$$S^+_\omega = \{s \in S \mid s \models \ell(\omega)\} \text{ and } S^-_\omega = \{s \in S \mid s \models \overline{\ell(\omega)}\}. \tag{3}$$

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
- $T = a \circ T'$, where $a \in A$, T' is a transition tree, and $a \circ [] = a$; 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. 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 $T = []$ then $\widehat{\Phi}([], S) = \{S\}$; else
- If $S = \perp$ or $S = \emptyset \wedge T \neq []$ then $\widehat{\Phi}(T, S) = \perp$; else
- If $T = a$, $a \in A$, then $\widehat{\Phi}(a, S) = \{\Phi(a, S)\}$; else
- If $T = a \circ T'$, $a \in A$, then $\widehat{\Phi}(T, S) = \widehat{\Phi}(T', \Phi(a, S))$; else

- If $T = \omega(T^+|T^-)$ then $\widehat{\Phi}(T, S) = \widehat{\Phi}(T^+, S^+_\omega) \cup \widehat{\Phi}(T^-, S^-_\omega)$ if $S \models pre(\omega)$, and $\widehat{\Phi}(T, S) = \perp$ otherwise.

Note that the definition of $\widehat{\Phi}$ allows the application of an observation ω in a belief state S where $\ell(\omega)$ is known. In this case, either $S^+_\omega = \emptyset$ or $S^-_\omega = \emptyset$ and the subtree rooted at the resulting empty belief state must be empty[2].

Let S_I be the initial belief state, i.e., the set of all possible states satisfying 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.

2.2. AND/OR forward Search for Contingent Planning Solutions

An *AND/OR search graph* T_s for a contingent planning problem $\mathcal{P}$, or *search graph* for short, is a labeled transition graph $\langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ where

- $\mathcal{S}$ is a set of belief states, each belief state is referred to as a node;
- s_0 is the initial belief state (the set of states that satisfy I) and $s_0 \in \mathcal{S}$; and
- $\mathcal{T}$ is a set of transitions, $\mathcal{T} \subseteq \mathcal{S} \times (A \cup \Omega) \times \mathcal{S}$, such that
  - for each $(s, \omega, s_1) \in \mathcal{T}$ such that $\omega \in \Omega$, $s \models pre(\omega)$ and there exists exactly one $(s, \omega, s_2) \in \mathcal{T}$ such that $\{s_1, s_2\} = \{s^+_\omega, s^-_\omega\}$; and
  - for each $(s, a, s_1) \in \mathcal{T}$ such that $a \in A$, $s \models pre(a)$ and $s_1 = \Phi(a, s)$.

Intuitively, a search graph represents the transition graph that have been expanded so far during the search. Each node encodes a belief state, the initial belief state s_0 is the start node, and $\mathcal{T}$ is the set of transitions between nodes (belief states). A transition (s, t, s') is called an *or-edge* (resp. *and-edge*) if $t \in A$ (resp. $t \in \Omega$) and s' is called an *or-child* (resp. *and-child*) of s. Two and-edges from the same node of the same observation are called *dual*. For convenience, we often refer to a transition $(s, t, s') \in T$ as an *outgoing edge* from s (or an *incoming edge* to s'). A *complete transition* is either an or-edge or a pair of dual and-edges. A *leaf node* is a node that has no children. There are no outgoing edges from a leaf node. A search graph is in general not a tree as a node may have multiple incoming edges. A graph $\langle \mathcal{S}_s, \mathcal{T}_s, s \rangle$ is a subgraph of a search graph $\langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ if $\mathcal{S}_s \subseteq \mathcal{S}$, $\mathcal{T}_s \subseteq \mathcal{T}$, and $s \in \mathcal{S}_s$.

A subgraph $\langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ of an AND/OR search graph $\langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ for $\mathcal{P}$ is called a *solution tree for* $\mathcal{P}$ if:

1. Every leaf node in $\mathcal{S}_g$ satisfies the goal G,
2. From each non-leaf node $s \in \mathcal{S}_g$ there exists either exactly one outgoing or-edge $(s, a, s') \in \mathcal{T}_g$ and $s' \in \mathcal{S}_g$ (and hence, s has exactly one or-child $s' \in \mathcal{S}_g$) or exactly a pair of outgoing dual and-edges $(s, \omega, s_1), (s, \omega, s_2) \in \mathcal{T}_g$ and $s_1, s_2 \in \mathcal{S}_g$ (and hence, s has exactly a pair of dual and-children $s_1, s_2 \in \mathcal{S}_g$), and
3. The subgraph does not contain a loop, i.e., $\mathcal{T}_g$ does not contain a set of edges of the form $\{(s_1, t_1, s_2), (s_2, t_2, s_3), \ldots, (s_n, t_n, s_1)\}$, where $n \geq 1$.

For each node s in a solution tree $\langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ for $\mathcal{P}$, by $tree(\mathcal{S}_g, \mathcal{T}_g, s)$ we denote the following transition tree:

- $tree(\mathcal{S}_g, \mathcal{T}_g, s) = []$ if $s \models G$;

[2] In the implementation, the application of an observation ω in belief state S is allowed only if $\ell(\omega)$ is unknown in S

- $tree(\mathcal{S}_g, \mathcal{T}_g, s) = a \circ tree(\mathcal{S}_g, \mathcal{T}_g, s')$ if $(s, a, s') \in \mathcal{T}_g$ and $a \in A$; and
- $tree(\mathcal{S}_g, \mathcal{T}_g, s) = \omega(tree(s_1)|tree(s_2))$ if $(s, \omega, s_1), (s, \omega, s_2) \in \mathcal{T}_g$ and $\omega \in \Omega$.

A node is newly added to the search graph is called *unexplored*. When a node is chosen for expansion, it becomes *explored*. We assume that when a node s is being explored, every possible outgoing edges from s is added to $\mathcal{T}$ and every possible child of s is added to $\mathcal{S}$. We also assume that a node that satisfies the goal is never explored as expanding the search graph through this node is useless. The search graph for $\mathcal{P}$ obtained when every node unsatisfying the goal has been explored is called the *complete search graph* for $\mathcal{P}$.

Theorem 1. *Let $T_s = \langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ be an AND/OR search graph for a contingent planning problem $\mathcal{P}$.*

1. *Let $T_g = \langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ be a subgraph of T_s. If T_g is a solution tree for $\mathcal{P}$ then $tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ is a solution for $\mathcal{P}$.*

2. *If T_s is the complete AND/OR search graph for $\mathcal{P}$ and $\mathcal{P}$ has a solution then there exists a subgraph of $\langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ that is a solution tree for $\mathcal{P}$.*

Proof. (Sketch) The proof for each statement is as follows

1. Since $\langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ is a solution tree, it does not contain a loop. By induction on the height of $\langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ (the longest path from s_0 to a leaf node), one can easily prove that $\widehat{\Phi}(tree(\mathcal{S}_g, \mathcal{T}_g, s_0), s_0)$ is the set of leaf nodes of $\langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$. Hence, every belief state in $\widehat{\Phi}(tree(\mathcal{S}_g, \mathcal{T}_g, s_0), s_0)$ satisfies the goal G. Moreover, $tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ is finite as a solution tree does not contain a loop. By definition, $tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ is a solution for $\mathcal{P}$.

2. Let T be a solution for $\mathcal{P}$. By induction on the height of T (the longest loop-free branch of the transition tree T) one can easily construct a solution tree $T_g = \langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ for $\mathcal{P}$ such that $tree(\mathcal{S}_g, \mathcal{T}_g, s_0) = T$. Since T_s is complete, T_g must be a subgraph of T_s (proof).

□

Theorem 1 allows for the use of an AND/OR forward search as a sound and complete algorithm for solving contingent planning problems.

Example 2. *Consider a simple problem $\mathcal{P}$ with an agent A and a bug B in a house with two rooms. Initially, B is healthy ($\neg dead(B) \wedge \neg wounded(B)$) but its location is unknown among the two rooms. The goal is B being dead ($dead(B)$). A can perform the following actions: move around within the two rooms ($move$), sense to determine whether B is in the same room with A ($sense(B)$), and kill B if they are in the same room ($kill(B)$). When A kills B, B will be dead if B is wounded and B can be either dead or wounded if it is healthy. Thus, the action $kill(B)$ is non-deterministic with the two outcomes $\{\emptyset \rightarrow dead(B) \wedge \neg wounded(B)\}$ and $\{\neg dead(B) \wedge \neg wounded(B) \rightarrow wounded(B), wounded(B) \rightarrow dead(B) \wedge \neg wounded(B)\}\}$). The problem $\mathcal{P}$ is given as*

- *Propositions:* $F = \{sameRoom, dead(B), wounded(B)\}$
- *Actions:* $A = \{move,\ kill(B)\}$, *where*
  - $pre(move) = \emptyset$, $O(move) = \{\{sameRoom \rightarrow \neg sameRoom, \neg sameRoom \rightarrow sameRoom\}\}$

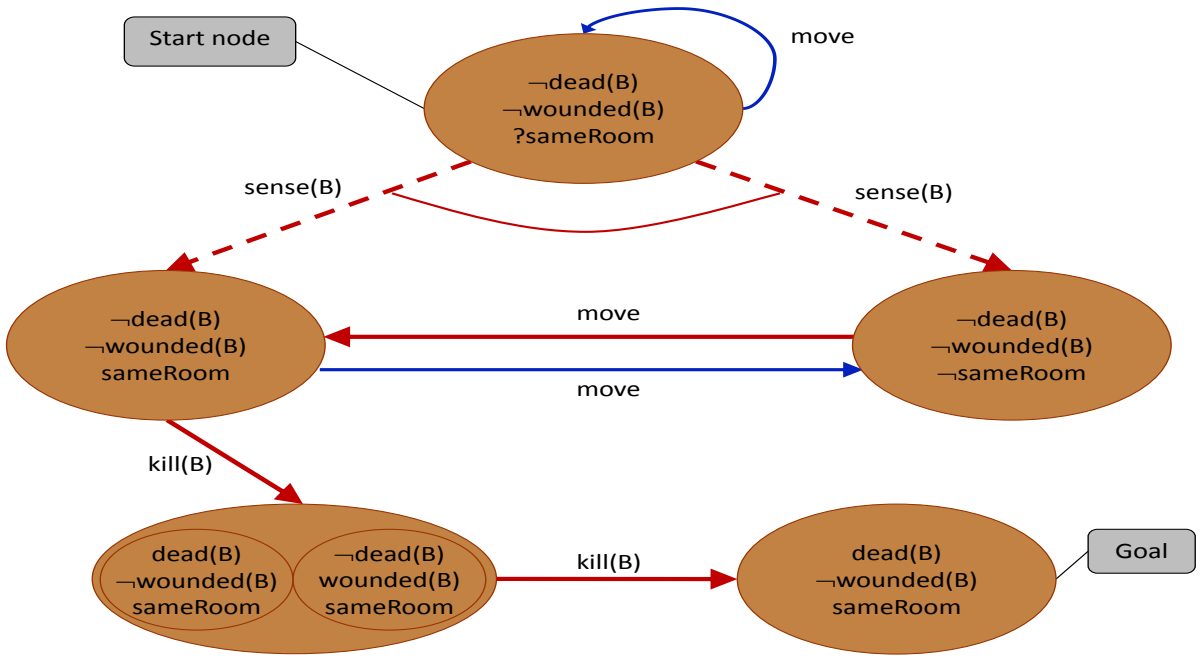

Figure 1: A contingent solution (red edges) in an AND/OR forward search graph. Solid edges denotes or-edges (actions) and dash edges denotes and-edges (observation). Two dual and-edges are connected by an arc.

  - $pre(kill(B)) = sameRoom$, $O(kill(B)) = \{\{\emptyset \rightarrow dead(B) \wedge \neg wounded(B)\}, \{\neg dead(B) \wedge \neg wounded(B) \rightarrow wounded(B), wounded(B) \rightarrow dead(B) \wedge \neg wounded(B)\}\}$

- *Observations:* $\Omega = \{sense(B)\}$, *where* $pre(sense(B)) = \emptyset$ *and* $\ell(sense(B)) = sameRoom$
- *Description of the initial state:* $I = \neg dead(B) \wedge \neg wounded(B)$
- *Goal:* $G = dead(B)$

Figure 1 demonstrates an AND/OR search graph for the problem $\mathcal{P}$*. The subgraph formed by the red edges and the nodes connected by those edges is a solution tree for* $\mathcal{P}$*. The corresponding solution for* $\mathcal{P}$ *is:*

$$"sense(B)(kill(B) \circ kill(B) \mid move \circ kill(B) \circ kill(B))"$$

3. Minimal-DNF Representation

This section reviews the *minimal-DNF* representation of belief states, adds relevant proofs (as they may not be provided in the previous work [18] due to lack of space), and presents an extension of the transition function proposed previously for conformant planning [18]. This extension is for handling non-deterministic and sensing actions required in contingent planning in addition to incomplete information.

3.1. Background

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. $|\Delta|$ denotes the number of partial states in Δ. The set $\Delta \setminus \{\delta \mid \exists \delta' \in \Delta . \delta' \subset \delta\}$, denoted by $\min(\Delta)$, 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. For a belief state S, we say that Δ *represents* S if $S = ext(\Delta)$.

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

$$\delta + \gamma = \begin{cases} \{\delta\} & \text{if } \gamma \subseteq \delta \text{ or } \overline{\gamma} \cap \delta \neq \emptyset \\ \{\delta \cup \gamma\} \cup \{\delta \cup \{\bar{\ell}\} \mid \ell \in \gamma \backslash \delta\} & \text{otherwise} \end{cases} \tag{4}$$

Proposition 1. *Let δ be a partial state and γ be a consistent set of literals. Then $\delta + \gamma$ is a DNF-state equivalent to δ and, for every $\delta' \in \delta + \gamma$, either $\delta' \models \gamma$ or $\delta' \models \neg\gamma$ holds.*

Proof. Consider two cases in Equation 4 as follows.

- If $\gamma \subseteq \delta$ or $\overline{\gamma} \cap \delta \neq \emptyset$, then $\delta + \gamma = \{\delta\}$. Obviously, this is a DNF-state equivalent to δ. Moreover, $\gamma \subseteq \delta$ iff $\delta \models \gamma$ and $\overline{\gamma} \cap \delta \neq \emptyset$ iff $\delta \models \neg\gamma$ (proof).
- In the other case, observe that $\delta \equiv (\delta \wedge \gamma) \vee (\delta \wedge \neg\gamma)$, where both $\delta \wedge \gamma$ and $\delta \wedge \neg\gamma$ are satisfiable. It is easy to see that $\delta \wedge \gamma \equiv \delta \cup \gamma$, a partial state (consistent set of literals). On the other hand, $\delta \wedge \neg\gamma \equiv \{\delta \cup \{\bar{\ell}\} \mid \ell \in \gamma\}$. Eliminating inconsistent sets of literals from $\{\delta \cup \{\bar{\ell}\} \mid \ell \in \gamma\}$, we obtain $\{\delta \cup \{\bar{\ell}\} \mid \ell \in \gamma \backslash \delta\}$, a set of partial states equivalent to $\delta \wedge \neg\gamma$. Observe that no partial states in $\{\delta \cup \{\bar{\ell}\} \mid \ell \in \gamma \backslash \delta\}$ is a subset of another one in this set as they all contain the same number of literals. Moreover, for each $\alpha \in \{\delta \cup \{\bar{\ell}\} \mid \ell \in \gamma \backslash \delta\}$, $\delta \cup \gamma$ cannot be either a superset or a subset of α as $\delta \cup \gamma \models \gamma$ and $\alpha \models \neg\gamma$. Thus, $\{\delta \cup \gamma\} \cup \{\delta \cup \{\bar{\ell}\} \mid \ell \in \gamma \backslash \delta\}$ is a DNF-state equivalent to δ and, for every partial state δ' in this DNF-state, either $\delta' \models \gamma$ or $\delta' \models \neg\gamma$ holds. In this case, since $\delta + \gamma = \{\delta \cup \gamma\} \cup \{\delta \cup \{\bar{\ell}\} \mid \ell \in \gamma \backslash \delta\}$, a proof is obtained.

In both cases, the proposition has been proved. □

For a DNF-state Δ, let $\Delta + \gamma = \min(\bigcup_{\delta \in \Delta} (\delta + \gamma))$.

3.2. Transition Function Φ_{DNF} for Non-deterministic Actions

This subsection presents the extension of the transition function defined in [18] to non-deterministic actions.

The definition of *enabling* notion is extended as follows

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

A DNF-state Δ is enabling *for o if every partial state in Δ is enabling for o.*

For an action outcome o and a DNF-state Δ, let $exp_o(\Delta) = ((\Delta + \psi_1) + \ldots) + \psi_k$, where $o = \{\psi_1 \rightarrow \ell_1, \ldots, \psi_k \rightarrow \ell_k\}$.

Proposition 2. *For every DNF-state Δ and every action outcome o, $\exp_o(\Delta)$ is a DNF-state equivalent to Δ and enabling for o.*

Proof. The proof is by induction on the number of conditional effects in the outcome o as follows.

- Base case $|o| = 1$: We have $exp_o(\Delta) = \Delta + \psi_1 = min(\bigcup_{\delta \in \Delta}(\delta + \psi_1))$, where $o = \{\psi_1 \rightarrow \ell_1\}$. Consider each partial state $\delta \in \Delta$, by Proposition 1, $\delta + \psi_1$ is a DNF-state equivalent to δ and either $\delta \models \psi_1$ or $\delta \models \neg\psi_1$ holds. This implies that $exp_o(\Delta)$ is a DNF-state equivalent to Δ and enabling for o (proof).

- Inductive step: Suppose the proposition is true for $|o| = k - 1$ for some $k > 1$. Consider an outcome o of k conditional effects, $o = \{\psi_1 \rightarrow \ell_1, \ldots, \psi_k \rightarrow \ell_k\}$. Let o' be the outcome of the first $k - 1$ conditional effects of o: $o' = \{\psi_1 \rightarrow \ell_1, \ldots, \psi_k \rightarrow \ell_{k-1}\}$. Then, by definition we have $exp_o(\Delta) = exp_{o'}(\Delta) + \psi_k$. By the inductive hypothesis, $exp_{o'}(\Delta)$ is a DNF-state equivalent to Δ and enabling for o'. Hence, as proved in the base case, $exp_{o'}(\Delta) + \psi_k$ is a DNF-state equivalent to $exp_{o'}(\Delta)$, which is equivalent to Δ. This implies that $exp_o(\Delta)$ is a DNF-state equivalent to Δ. Consider each partial state δ_1 in $exp_{o'}(\Delta) + \psi_k$. Observe from Equation 4 that δ_1 is a superset of some partial state δ_2 in $exp_{o'}(\Delta)$. Hence, for every ψ_i, $i = 1, \ldots, k-1$, if $\delta_2 \models \psi_i$ (resp. $\delta_2 \models \neg\psi_i$) then $\delta_1 \models \psi_i$ (resp. $\delta_1 \models \neg\psi_i$). This implies that, like δ_2, δ_1 is also enabling for o'. Moreover, since $\delta_1 \in exp_{o'}(\Delta) + \psi_k$, either $\delta_1 \models \psi_k$ or $\delta_1 \models \neg\psi_k$ holds. Thus, δ_1 is enabling for o and so is $exp_{o'}(\Delta) + \psi_k$. In conclusion, $exp_o(\Delta)$ is a DNF-state equivalent to Δ and enabling for o (proof).

□

For an outcome o of an action a and a partial state δ, the effect of a in δ if the outcome o occurs, denoted $e(o, \delta)$, is defined similarly the effect of a in a state s w.r.t. o:

$$e(o, \delta) = \{\ell \mid \psi \rightarrow \ell \in o, \delta \models \psi\}. \tag{5}$$

Proposition 3. *Let o be an action outcome and δ be a partial state enabling for o, then*

$$\forall s \in ext(\delta).\, e(o, s) = e(o, \delta)$$

Proof. Let s be an arbitrary state in $ext(\delta)$. It suffices to prove that $e(o, s) = e(o, \delta)$.

(1) For each $\ell \in e(o, \delta)$, we will show that $\ell \in e(o, s)$: by Equation 5, there exists $\psi \rightarrow \ell \in o$ such that $\delta \models \psi$. This implies $s \models \psi$, since $\delta \subseteq s$. Hence, $\ell \in e(o, s)$ (Equation 1)

(2) Now for each $\ell' \in e(o, s)$, we prove that $\ell' \in e(o, \delta)$: Assume that $\ell' \notin e(o, \delta)$. Since $\ell' \in e(o, s)$, there exists an effect $\psi' \rightarrow \ell' \in o$ such that $s \models \psi'$. On the other hand, $\ell' \notin e(o, \delta)$ so $\delta \not\models \psi'$. This means that $\delta \models \neg\psi'$ (because δ is enabling for o, either $\delta \models \psi'$ or $\delta \models \neg\psi'$ holds). This implies $s \models \neg\psi'$ as $\delta \subseteq s$, a contradiction.

The proof has been obtained by (1) and (2). □

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:*

$$\Phi_{DNF}(a,\Delta) = \begin{cases} min(\bigcup_{o \in O(a)} \{\delta' \setminus \overline{e(o,\delta')} \cup e(o,\delta') \mid \delta' \in exp_o(\Delta)\}) & \text{if } \Delta \neq \emptyset \wedge \Delta \models pre(a) \\ undefined & \text{otherwise} \end{cases} \tag{6}$$

Lemma 1. *Let Δ be a set of partial states and e be a consistent set of literals. Then,*

$$ext(\{\delta \setminus \bar{e} \cup e \mid \delta \in \Delta\}) = \{s \setminus \bar{e} \cup e \mid s \in ext(\Delta)\}$$

Proof. First we will prove that, for every partial state δ,

$$ext(\delta \setminus \bar{e} \cup e) = \{s \setminus \bar{e} \cup e \mid s \in ext(\delta)\} \tag{7}$$

Consider a state $s \in ext(\delta)$, we have $\delta \subseteq s$. Hence, $\delta \setminus \bar{e} \cup e \subseteq s \setminus \bar{e} \cup e$, i.e., $s \setminus \bar{e} \cup e \models \delta \setminus \bar{e} \cup e$ or $s \setminus \bar{e} \cup e \in ext(\delta \setminus \bar{e} \cup e)$ (it is easy to see that $s \setminus \bar{e} \cup e$ is a state). This means that

$$\{s \setminus \bar{e} \cup e \mid s \in ext(\delta)\} \subseteq ext(\delta \setminus \bar{e} \cup e) \tag{8}$$

Now, let s_1 be an arbitrary state in $ext(\delta \setminus \bar{e} \cup e)$, we will prove that s_1 also belongs to $\{s \setminus \bar{e} \cup e \mid s \in ext(\delta)\}$. Since $s_1 \in ext(\delta \setminus \bar{e} \cup e)$, $s_1 \models (\delta \setminus \bar{e} \cup e)$ or $(\delta \setminus \bar{e} \cup e) \subseteq s_1$. Let $\gamma = s_1 \setminus (\delta \setminus \bar{e} \cup e)$. Observe that γ is a consistent set of literals that are *independent* from $\delta \setminus \bar{e} \cup e$ (we say that a literal ℓ is independent from a formula φ if neither ℓ nor $\bar{\ell}$ appears in φ). Let $e^- = \delta \cap \bar{e}$, $e^+ = e \setminus \overline{e^-}$, and $\delta_0 = \delta \setminus e^-$. Observe that, $s_1 = \delta \setminus \bar{e} \cup e \cup \gamma = (\delta_0 \cup e^-) \setminus (e^- \cup \overline{e^+}) \cup (\overline{e^-} \cup e^+) \cup \gamma = \delta_0 \cup \overline{e^-} \cup e^+ \cup \gamma$. Now consider $s_2 = \delta_0 \cup e^- \cup e + \cup \gamma = \delta \cup e^+ \cup \gamma$. Furthermore, every partition set in s_2: δ_0, e^-, e^+, and γ are consistent; e^- and γ are independent from all the other sets while $\delta_0 \cup e^+$ is consistent. Therefore, s_2 is consistent and thereby it is a state (because the states s_1 and s_2 are complete). Since $\delta \subseteq s_2$ so $s_2 \in ext(\delta)$. Observe that $s_2 \setminus \bar{e} \cup e = \delta \cup e^+ \cup \gamma \setminus \bar{e} \cup e = \delta \setminus \bar{e} \cup e \cup \gamma = s_1$ (because γ is independent from $\bar{e}$ and $e^+ \subseteq e$). This means that $s_1 \in \{s \setminus \bar{e} \cup e \mid s \in ext(\delta)\}$. This implies that

$$ext(\delta \setminus \bar{e} \cup e) \subseteq \{s \setminus \bar{e} \cup e \mid s \in ext(\delta)\} \tag{9}$$

Together (8) and (9) we have the proof for (7). Now we have

$$\begin{aligned} ext(\{\delta \setminus \bar{e} \cup e \mid \delta \in \Delta\}) &= \bigcup_{\delta \in \Delta} ext(\delta \setminus \bar{e} \cup e) \\ &= \bigcup_{\delta \in \Delta} \{s \setminus \bar{e} \cup e \mid s \in \delta\} && \text{(By Equation (7))} \\ &= \{s \setminus \bar{e} \cup e \mid s \in \bigcup_{\delta \in \Delta} ext(\delta)\} \\ &= \{s \setminus \bar{e} \cup e \mid s \in ext(\Delta)\} && \text{(Proof)} \end{aligned}$$

□

Theorem 2. *Let Δ be a DNF-state and a be an action. Then,*

$$ext(\Phi_{DNF}(a, \Delta)) = \Phi(a, ext(\Delta))$$

Proof. It is easy to see that $\Delta = \emptyset$ iff $ext(\Delta) = \emptyset$ and $\Delta \models pre(a)$ iff $ext(\Delta) \models pre(a)$. The second case is trivial. Now we are going to prove for the first case ($\Delta \neq \emptyset \wedge \Delta \models pre(a)$) as follows

$$
\begin{aligned}
ext(\Phi_{DNF}(a, \Delta)) &= ext(min(\bigcup_{o \in O(a)} \{\delta' \setminus \overline{e(o, \delta')} \cup e(o, \delta') \mid \delta' \in exp_o(\Delta)\})) \\
&= \bigcup_{o \in O(a)} ext(min(\{\delta' \setminus \overline{e(o, \delta')} \cup e(o, \delta') \mid \delta' \in exp_o(\Delta)\})) \\
&= \bigcup_{o \in O(a)} ext(\{\delta' \setminus \overline{e(o, \delta')} \cup e(o, \delta') \mid \delta' \in exp_o(\Delta)\}) \\
&\text{(Since for two partial states } \delta_1 \text{ and } \delta_2, \delta_1 \subset \delta_2 \text{ iff } ext(\delta_2) \subset ext(\delta_1)) \\
&= \bigcup_{o \in O(a)} \{s \setminus \overline{e(o, \delta')} \cup e(o, \delta') \mid s \in ext(exp_o(\Delta))\} && \text{(By Lemma 1)} \\
&= \bigcup_{o \in O(a)} \{s \setminus \overline{e(o, s)} \cup e(o, s) \mid s \in ext(exp_o(\Delta))\} && \text{(By Prop. 3)} \\
&= \bigcup_{o \in O(a)} \{s \setminus \overline{e(o, s)} \cup e(o, s) \mid s \in ext(\Delta)\} && \text{(By Prop. 2)} \\
&= \Phi(a, ext(\Delta)) && \text{(Proof)}
\end{aligned}
$$

□

According to Theorem 2, if a DNF-state Δ represents a belief state S ($ext(\Delta) = S$) then the successor DNF-state of Δ ($\Phi_{DNF}(a, \Delta)$) represents the successor belief state of S ($\Phi(a, S)$). This allows for building a sound and complete contingent planner using the minimal-DNF representation.

Lemma 2. *Let Δ be a DNF-state and o be an action outcome. Let n be the number of propositions in the domain, k be the number of distinct e-conditions of o that are unknown in Δ, and r be the maximum number of literals in an e-condition of o. Then $exp_o(\Delta)$ and $\{e(o, \delta) \mid \delta \in exp_o(\Delta)\}$ can be computed in $O(n|\Delta|^2(1 + r^2)^k)$ time and $|exp_o(\Delta)| \leq |\Delta|(1 + r)^k$.*

Proof. (Sketch) Observe that $exp_o(\Delta)$ is exactly the same as $exp_a(\Delta)$ defined in [18] for a deterministic action a, whose conditional effect set is o. Hence, to prove the lemma, we prove that $exp_a(\Delta)$ can be computed in $O(n|\Delta|^2(1 + r^2)^k)$ time and $|exp_a(\Delta)| \leq |\Delta|(1 + r)^k$. Due to the length and the complexity of the proof of this result, we omit it in this paper, which mainly focuses on the AND/OR search algorithm PrAO for contingent solutions. Interested readers are referred to a manuscript at `www.cs.nmsu.edu/~sto` submitted to Artificial Intelligence, where the proof can be found in the subsection Computational Cost of $\Phi_{\mu DNF}$ of the section Minimal DNF Representation. □

The following theorem shows the computational cost of Φ_{DNF}.

Theorem 3. *Let Δ be a DNF-state and a be an action. Let n be the number of propositions in the domain, k be the maximum number of distinct e-conditions in an outcome o of $O(a)$ that are unknown in Δ, and r be the maximum number of literals in an e-condition of $O(a)$. Then $\Phi_{DNF}(a, \Delta)$ can be computed in $O(n|\Delta|^2|O(a)|^2(1 + r)^{2k})$ time.*

Proof. We are interested only in the first case ($\Phi_{DNF}(a,\Delta) = min(\bigcup_{o\in O(a)}\{\delta' \setminus \overline{e(o,\delta')} \cup e(o,\delta') \mid \delta' \in exp_o(\Delta)\})$) as the proof for the second case is trivial. By Lemma 2, for each outcome o in $O(a)$, $exp_o(\Delta)$ and $exp_o(\Delta)$ and $\{e(o,\delta) \mid \delta \in exp_o(\Delta)\}$ can be computed in $O(n|\Delta|^2(1+r^2)^k)$ time and $|exp_o(\Delta)| \leq |\Delta|(1+r)^k$. Hence, the computation of all δ' and $e(o,\delta')$ in $\bigcup_{o\in O(a)}\{\delta' \setminus \overline{e(o,\delta')} \cup e(o,\delta') \mid \delta' \in exp_o(\Delta)\}$ is $O(n|\Delta|^2(1+r^2)^k|O(a)|)$ time and the number of partial states in the set is $O(|\Delta|(1+r)^k|O(a)|)$. Moreover, it is easy to see that for each partial state δ and for each set of literal e, $\delta \setminus \overline{e} \cup e$ can be computed in $O(n)$ time. Hence, $\bigcup_{o\in O(a)}\{\delta' \setminus \overline{e(o,\delta')} \cup e(o,\delta') \mid \delta' \in exp_o(\Delta)\}$ can be computed in $O(n|\Delta|^2(1+r^2)^k|O(a)|) + O(n|\Delta|(1+r)^k|O(a)|) = O(n|\Delta|^2(1+r^2)^k|O(a)|)$. Comparison of two partial states takes $O(n)$ time (assume that the literals in a partial state are sorted) so the application of the min function to the set $\bigcup_{o\in O(a)}\{\delta' \setminus \overline{e(o,\delta')} \cup e(o,\delta') \mid \delta' \in exp_o(\Delta)\}$ of $O(|\Delta|(1+r)^k|O(a)|)$ partial states takes $O(n|\Delta|^2(1+r)^{2k}|O(a)|^2)$ time. In summary, $\Phi_{DNF}(a,\Delta) = min(\bigcup_{o\in O(a)}\{\delta' \setminus \overline{e(o,\delta')} \cup e(o,\delta') \mid \delta' \in exp_o(\Delta)\})$ can be computed in $O(n|\Delta|^2(1+r^2)^k|O(a)|) + O(n|\Delta|^2(1+r)^{2k}|O(a)|^2) = O(n|\Delta|^2|O(a)|^2(1+r)^{2k})$ time (proof). □

Theorem 3 shows that $\Phi_{DNF}(a,\Delta)$ is exponential in k with the base $r+1$, where k is the maximum number of distinct e-conditions in an outcome in $O(a)$ that are unknown in Δ and r is the maximum number of literals in an e-condition of a conditional effect of the action. In most problems in the literature, r is 1 or 2 while k is rather small in many problems. Thus, $\Phi_{DNF}(a,\Delta)$ can be computed in polynomial time on many problems.

3.3. Extension to Sensing Actions

Let ω be a sensing action and Δ be a DNF-state. Similar to that for belief states (Equation (3)), the result of the execution of ω in Δ is a pair of DNF-states, one satisfies $\ell(\omega)$ and the other satisfies $\overline{\ell(\omega)}$ and their union is equivalent to Δ. To compute the pair of DNF-states, we need to extend Δ w.r.t. $\ell(\omega)$ so that for each partial state δ in the new DNF-state, either $\delta \models \ell(\omega)$ or $\delta \models \overline{\ell(\omega)}$ holds. The pair of DNF-states is obtained by splitting the new DNF-state based on $\ell(\omega)$.

Definition 3. *Let ω be a sensing action and Δ be a DNF-state. The application of ω in Δ results in a pair of sets of partial states, denoted by Δ^+_ω and Δ^-_ω, defined as*

$$\Delta^+_\omega = \{\delta \in \Delta + \{\ell(\omega)\} \mid \delta \models \ell(\omega)\}$$

$$\Delta^-_\omega = \{\delta \in \Delta + \{\ell(\omega)\} \mid \delta \models \overline{\ell(\omega)}\}.$$

Theorem 4. *Let Δ be a DNF-state representing a belief state S and ω be an observation. Then Δ^+_ω is a DNF-state representing S^+_ω, and Δ^-_ω is a DNF-state representing S^-_ω.*

Proof. Observe that a subset of a DNF-state is also a DNF-state as it is a set of partial states such that no one is a subset of another. Hence, Δ^+_ω and Δ^-_ω are DNF-states as they are subsets of the DNF-state $\Delta + \{\ell(\omega)\}$. Now we need to prove $ext(\Delta^+_\omega) = S^+_\omega$ and $ext(\Delta^-_\omega) = S^-_\omega$. Observe that $\Delta + \{\ell(\omega)\} = \Delta^+_\omega \cup \Delta^-_\omega$. Hence, $ext(\Delta^+_\omega) \cup ext(\Delta^-_\omega) = ext(\Delta + \{\ell(\omega)\}) = ext(\Delta) = S$. By definition, $\Delta^+_\omega \models \ell(\omega)$ and $\Delta^-_\omega \models \overline{\ell(\omega)}$. This implies that $ext(\Delta^+_\omega) = \{s \in S \mid s \models \ell(\omega)\}$ and $ext(\Delta^-_\omega) = \{s \in S \mid s \models \overline{\ell(\omega)}\}$, since $ext(\Delta^+_\omega) \cup ext(\Delta^-_\omega) = S$ and $ext(\Delta^+_\omega) \cap ext(\Delta^-_\omega) = \emptyset$. Thus, $ext(\Delta^+_\omega) = S^+_\omega$ and $ext(\Delta^-_\omega) = S^-_\omega$ (proof). □

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

Proof. (Sketch) A proof by induction on the depth of the transition tree T can be easily obtained using Theorems 2 and 4. □

This theorem allows for the development of contingent planners that employ an AND/OR forward search in the belief space where each node in the search graph is a DNF-state instead of a belief state.

4. A Standard Heuristic AND/OR forward Search Algorithm for Contingent Planning

Let $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$ be a contingent planning problem. For completeness of the paper, this section presents a standard heuristic AND/OR forward search algorithm for a solution of $\mathcal{P}$ and the correctness of the algorithm as the foundation of the PrAO algorithm.

Let s_0 be the initial belief state for the problem $\mathcal{P}$. The search iteratively expands the search graph $T_s = \langle \mathcal{S}, \mathcal{T}, s_0\rangle$, initialized with $\langle\{s_0\}, \emptyset, s_0\rangle$, and updates the state of nodes in the graph. Initially the start node s_0 is set as unexplored (if it does not satisfies the goal G) and in each iteration the search explores an unexplored node in $\mathcal{S}$ until a solution is detected or it is determined that $\mathcal{P}$ has no solution. The main procedure of this AND/OR search is implemented in Algorithm 1.

In the procedure $Plan(F, A, \Omega, I, G)$ (Algorithm 1) and its sub-procedures, the components of the problem $\mathcal{P}$ and the search graph $\langle \mathcal{S}, \mathcal{T}, s_0\rangle$ are represented by global variables. For each node $s \in \mathcal{S}$, the variable $state(s)$ is used to encode the state of s, which can be $explored$, $unexplored$, $goal$, or $dead$. The state $goal$ or $dead$ indicates that the node is *goal-reachable* or *dead-end*, respectively, and will be defined next.

Definition 4. *Let $\langle \mathcal{S}, \mathcal{T}, s_0\rangle$ be a search graph for $\mathcal{P}$. A node s is* goal-reachable*, or* goal *for short, if*

- *$s \models G$,*
- *There exists an or-edge (s, a, s_1) in $\mathcal{T}$ such that s_1 is goal-reachable, or*
- *There exist two dual and-edges (s, ω, s_1) and (s, ω, s_2) in $\mathcal{T}$ such that s_1 and s_2 are goal-reachable.*

For each goal node s, there exists a path from s to a node s' such that $s' \models G$ and every node on the path is a goal node. By *path-to-goal* we call such a path where any possible loops are eliminated. By *distance-to-goal of s* we refer to the number of edges of the longest path-to-goal of s.

Proposition 4. *If a node s on a search graph $T_s = \langle \mathcal{S}, \mathcal{T}, s_0\rangle$ for $\mathcal{P}$ is goal-reachable, then there exists a subgraph $\langle \mathcal{S}_s, \mathcal{T}_s, s\rangle$ of the search graph T_s that is a solution tree for the problem $\mathcal{P}_s = \langle F, A, \Omega, s, G\rangle$.*

Proof. The proof is by induction on the distance-to-goal of s.

- Base case, the distance-to-goal of s is 0: then $s \models G$. Clearly, the subgraph $\langle\{s\}, \emptyset, s\rangle$ is a solution tree for $\mathcal{P}_s$.

Algorithm 1 $Plan(F, A, \Omega, I, G)$ The main procedure of a standard AND/OR search algorithm for contingent planning

Input: A contingent problem $\mathcal{P} = \langle F, A, \Omega, I, G \rangle$
Output: A solution for $\mathcal{P}$ if it has a solution or $NULL$ otherwise
Let s_0 be the initial belief state of $\mathcal{P}$ {The set of states satisfying I}
if $s_0 \models G$ **then**
  return [] {Return the empty solution}
else
  Set $state(s_0) = unexplored$
end if
Let $\mathcal{S} = \{s_0\}, \mathcal{T} = \emptyset$ {Initialize the search graph with $(\{s_0\}, \emptyset, s_0)$}
while true do
  if $\{s \in \mathcal{S} \mid state(s) = unexplored\} = \emptyset$ **then**
    return $NULL$ {No solutions}
  end if
  Let s be the unexplored node with the best heuristic in $\mathcal{S}$
  $explore(s)$ {Expand the search graph and update the state of nodes by exploring the node s}
  if $state(s_0) = goal$ **then**
    $solution\_tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ {Extract a solution tree $\langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ from $\langle \mathcal{S}, \mathcal{T}, s_0 \rangle$}
    return $tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ {Return a solution for the problem $\mathcal{P}$}
  end if
  if $state(s_0) = dead$ **then**
    return $NULL$ {No solutions}
  end if
end while

- Inductive step: Assume that the proposition is true for every goal node with the distance-to-goal not greater than d, for some $d \geq 0$. Consider a case where the distance-to-goal of s is $d + 1$. Since s is goal-reachable, we consider the following two cases either one of which holds.
  - There exists an or-edge (s, a, s') in $\mathcal{T}$ and s' is a goal node. By the inductive hypothesis, there exists a subgraph $\langle \mathcal{S}_{s'}, \mathcal{T}_{s'}, s' \rangle$ of T_s that is a solution tree for $\mathcal{P}_{s'} = \langle F, A, \Omega, s', G \rangle$. Let $\mathcal{S}_s = \mathcal{S}_{s'} \cup \{s\}$ and $\mathcal{T}_s = \mathcal{T}_{s'} \cup \{(s, a, s')\}$. Obviously, $\langle \mathcal{S}_s, \mathcal{T}_s, s \rangle$ is a subgraph of T_s. Moreover, by the definition of a solution tree, it is easy to see that $\langle \mathcal{S}_s, \mathcal{T}_s, s \rangle$ is a solution tree for the problem $\mathcal{P}_s = \langle F, A, \Omega, s, G \rangle$.
  - There exist two dual and-edge (s, ω, s_1) and (s, ω, s_2) in $\mathcal{T}$ such that both s_1 and s_2 are goal-reachable. The proof is similar to the previous case.

□

Intuitively, every node in a solution tree is goal-reachable and if the start node s_0 becomes goal-reachable then the search tree contains a solution tree as shown in the following theorem.

Theorem 6. *If the start node s_0 is a goal-reachable node on a search graph $T_s = \langle \mathcal{S}, \mathcal{T}, s_0 \rangle$, then T_s contains a solution tree for $\mathcal{P}$.*

Proof. This proposition is a direct corollary of Proposition 4. □

Theorem 6 is instantiated in Lines 15-18 of Algorithm 1, i.e., if the start node s_0 becomes goal-reachable then the procedure $solution\_tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ extracts a solution tree $\langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ from the search graph T_s and from the obtained solution tree it computes and returns the corresponding solution $tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$. One may think that the extraction of a solution tree from a search graph when it is detected to contain a solution tree is simple. Indeed, since from a goal node there can be multiple outgoing transitions that lead to other goal nodes and there may exists a loop in a subgraph formed by a set of goal nodes, this computation is rather complicated and can be computationally exponential in the number of transitions from a goal node to other goal nodes with the base equal to the number of goal nodes as it may incurs backtracking. Since the purpose of presenting this standard AND/OR search algorithm is to provide a foundation for the development of the PrAO algorithm, we do not present an algorithm for the extraction of a solution tree. Instead, we will show in the next section that when PrAO detects a solution, due to the pruning techniques it employs, the remaining search graph is also a solution tree for the contingent planing problem. This demonstrates another desirable property of PrAO besides the reduction of the search space.

The dead-end ($dead$) state of a node is defined as follows.

Definition 5. *Let $\langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ be a search graph for $\mathcal{P}$. An explored node s is* dead-end*, or* dead *for short, if*

- *$s \not\models G$ and there is no outgoing edge 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.*

For a node s, by the *depth of s* we refer to the number of edges on the longest loop-free path from s to a leaf node.

Proposition 5. *Let $\langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ be a search graph for $\mathcal{P}$ and s be a dead node. Then there does not exists a solution tree for $\mathcal{P}$ that contains s.*

Proof. To prove the proposition, we prove that s is not a goal node and can never become a goal node when the search graph is expanded further. The proof is by induction on the depth d of s.

- Base case, the depth of s is 0: Clearly s cannot be a goal node since $s \not\models G$. Since s is dead, s is an explored node and there is no outgoing edge from s. Hence, s can never become a goal node as expanding the search graph can only create new edges from unexplored nodes.

- Inductive step: Suppose that the proposition is true for every node with depth less than d, for some $d \geq 0$. We will show that it also true for every node with depth $d+1$. Consider a dead node s with the depth $d+1$. By definition, s is not a goal node in the current search graph. s is a node in a solution tree only if s is a goal node. This can happens only if a child of s becomes goal-reachable. By the inductive hypothesis, every dead child of s can never become a goal node. The only children of s can possibly become goal-reachable are and-children of s. However, the fact that those nodes become goal-reachable cannot make s goal-reachable since their dual and-nodes are dead nodes and can never become a goal node. Thus, s can never be a goal node.

Since s can never be a goal node, there does not exists a solution tree for $\mathcal{P}$ that contains s (proof). □

Theorem 7. *If the start node s_0 is a dead node on a search tree then $\mathcal{P}$ has no solution.*

Proof. The proof follows directly from Proposition 5. □

One can see that Theorem 7 is instantiated in the main procedure by Lines 19-21, Algorithm 1. In each iteration of the **while** loop, the $explore(s)$ procedure (Line 14) expands the search graph and updates the state of their nodes accordingly by exploring the unexplored node s with the best heuristic value. This procedure is implemented in Algorithm 2.

Algorithm 2 $explore(s)$ Expand and update the search graph by exploring an unexplored node s

1: **Input**: (unexplored) node $s \in \mathcal{S}$
2: **for** each action $a \in A$ **do**
3: **if** $s \models pre(a)$ **then**
4: Compute $s' = \Phi(a, s)$ {Compute a successor (or-child) of s}
5: **if** $s' \neq s$ **then**
6: $expand(s, a, s')$ {Extend the graph with (only new) node s' and edge (s, a, s'), Algorithm 3}
7: $goal\_propagation(s, a, s')$ {Algorithm 4}
8: **end if**
9: **end if**
10: **end for**
11: **for** each observation $\omega \in \Omega$ **do**
12: **if** $s \models pre(\omega)$ **and** $s \not\models \ell(\omega)$ **and** $s \not\models \overline{\ell(\omega)}$ **then**
13: Compute $s_1 = s^+_\omega$, $s_2 = s^-_\omega$ {Compute a pair of dual and-children of s}
14: $expand(s, \omega, s_1)$ {Extend the graph with node s_1, if it is a new node, & and-edge (s, ω, s_1)}
15: $expand(s, \omega, s_2)$ {Extend the graph with node s_2, if it is a new node, & and-edge (s, ω, s_2)}
16: $goal\_propagation(s, \omega, s_1)$ {Algorithm 4}
17: **end if**
18: **end for**
19: **if** $\exists(s, t, s') \in \mathcal{T}$ **then**
20: Set $state(s) = explored$
21: **else**
22: Set $state(s) = dead$ {The explored node s is dead if there are no outgoing transitions from s}
23: $dead\_propagation(s)$ {Algorithm 5}
24: **end if**

Algorithm 3 $expand(s, t, s')$ Set the state of s' and extend the graph with node s' (if s' is a new node) and edge (s, t, s')

1: **Input**: node s, $t \in A \cup \Omega$, node s'
2: **if** $s' \notin \mathcal{S}$ **then**
3: **if** $s' \not\models G$ **then**
4: Set $state(s') = unexplored$
5: **else**
6: Set $state(s') = goal$
7: **end if**
8: $\mathcal{S} = \mathcal{S} \cup \{s'\}$ {Extend the graph with the new node s'}
9: **end if**
10: $\mathcal{T} = \mathcal{T} \cup \{(s, t, s')\}$ {Extend the graph with the edge (s, t, s')}

During the exploration of a node s, s can become goal-reachable if it has a goal or-child or a pair of goal dual and-children (which can be either a new node satisfying the goal or an existing goal node). In turn, the parent(s) of s, and hence some of its ancestors, may become goal-reachable if s becomes goal-reachable. The $goal\_propagation(s, t, s')$ procedure (Algorithm 4) recursively identifies the nodes that become goal due to the addition of an edge (s, t, s') to the graph (if s' is a goal node) and updates the state of those nodes as goal (Lines 7 & 16, Algorithm 2). Observe that Line 5 eliminates the creation of an edge from a node to the same node. A similar condition for and-transitions is not needed due to the conditions $s \not\models \ell(\omega)$ and $s \not\models \overline{\ell(\omega)}$ (Line 12), that make sure $s \neq s_1$ and $s \neq s_2$.

Algorithm 4 $goal\_propagation(s, t, s_1)$

1: **Input**: A transition $(s, t, s_1) \in \mathcal{T}$
2: **if** $state(s_1) \neq goal$ **or** $state(s) = goal$ **then**
3: **return**
4: **end if**
5: **if** $t \in A$ **then**
6: Set $state(s) = goal$
7: **for** each $(s', t', s) \in \mathcal{T}$ **do**
8: $goal\_propagation((s', t', s))$
9: **end for**
10: **else**
11: Let (s, t, s_2) be the dual and-edge of (s, t, s_1) in $\mathcal{T}$
12: **if** $state(s_2) = goal$ **then**
13: Set $state(s) = goal$
14: **for** $(s', t', s) \in \mathcal{T}$ **do**
15: $goal\_propagation((s', t', s))$
16: **end for**
17: **end if**
18: **end if**

The following proposition validates the correctness of Algorithm 4.

Proposition 6. *Let s be the node being explored and (s, t, s_1) be a new edge added to the search graph. Suppose that the set $\{s \in \mathcal{S} \mid state(s) = goal\}$ is equal to the set of goal-reachable nodes of the search graph before the edge (s, t, s_1) is added to $\mathcal{T}$. Then, after (s, t, s_1) is added to $\mathcal{T}$ and the execution of $goal\_propagation(s, t, s_1)$ is completed, the two sets remain equal.*

Proof. (Sketch) It is easy to see that $goal\_propagation$ changes the state of a node to be $goal$ only if the node is goal-reachable. Hence, after an edge (s, t, s_1) is added to $\mathcal{T}$ and the execution of $goal\_propagation(s, t, s_1)$ is completed, every node s with $state(s) = goal$ on the search graph is indeed goal-reachable. Now we need to prove that there does not exist a node s' such that s' is goal-reachable but $state(s') \neq goal$. Assume the contrary, i.e., there exists such a node s'. Due to the precondition ($\{s \in \mathcal{S} \mid state(s) = goal\}$ is exactly the set of goal-reachable nodes before the edge (s, t, s_1) is added to $\mathcal{T}$) and the fact that $goal\_propagation$ only changes the state of a node to be $goal$, s' is goal-reachable due to its goal-reachable descendant s_1 but $state(s')$ is not changed to be $goal$ by $goal\_propagation$. This means that the propagation process stops at a node s_2 before s' on the reverse path from s_1 to s', where s_2 is an ancestor of s_1. However, this propagation stops at s_2 only if:

- s_2 is already a goal node before the propagation (Line 2). In this case, a propagation from s_2 was already executed which updated the state of s' correctly. The further propagation towards s' this time if executed would only repeat what was done before and would not change the state of s. In other words, if $state(s')$ is changed to $goal$ by the propagation this time if it continued until s', then $state(s')$ would have been changed to $goal$ by the earlier propagation when its descendant s_2 became goal-reachable. Thus, this case cannot make $state(s')$ to be wrong.
- The child of s_2 on the path to s_1 is not goal-reachable (Line 2). Look into Algorithm 4, this is the case only if s_1 is not a goal node. However, the earlier discussion shows that s_1 must be goal-reachable.
- The child of s_2 on the path to s_1 is an and-child of s_2 and the other dual and-child is not goal-reachable (consider Lines 11-12). In this case, by definition, s_2 does not become goal-reachable and, hence, the termination of the propagation is correct. s' cannot become goal-reachable due to s_2 and thereby due to s_1.

In all the cases, we showed that the existence of such a node s' is impossible (proof). □

After a node s is explored, s may become a dead node (there does not exist an outgoing edge from s). If s is dead then some of its parent(s) can also become dead; and if a parent of s becomes dead then, in turn, some of the parents of that parent of s can become dead and so on. Similar to $goal\_propagation$, the $dead\_propagation$ procedure (Algorithm 5) identifies the set of dead-end nodes and changes their state to $dead$.

Algorithm 5 $dead\_propagation(s)$

```
1: Input: A dead node s ∈ S
2: for each edge (s_1, t, s) ∈ T do
3:    {Check whether s_1 is dead-end according to Definition 5}
4:    if s_1 is dead-end then
5:       Set state(s_1) = dead
6:       Execute dead_propagation(s_1)
7:    end if
8: end for
```

The correctness of the $dead\_propagation$ procedure (Algorithm 5) is expressed in the following proposition.

Proposition 7. *Let s be a node being explored. Suppose that the set $\{s \in \mathcal{S} \mid state(s) = dead\}$ is equal to the set of dead-end nodes of the search graph before exploring s and there does not exist an outgoing edge from s after the exploration of s. Then, after the exploration of s and the execution of $dead\_propagation(s)$, the two sets remain equal.*

Proof. The proof follows directly from Definition 5. □

Proposition 8. *If a node ever becomes a goal node or a dead node in the search graph then its state can never change.*

Proof. This is obvious as a dead node cannot become an unexplored node (no explored node can become unexplored again), an explored node (a dead node will not be explored again), or a goal node. Similarly for a goal node. □

With the assumption that the procedure $solution\_tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ (Line 16, Algorithm 1) always returns a solution tree if the search graph contains a solution tree, the correctness of the presented search algorithm is confirmed by the following theorem.

Theorem 8. *Let $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$ be a contingent planning problem and $T_s = \langle \mathcal{S}, \mathcal{T}, s_0\rangle$ be a search graph constructed by the $Plan(F, A, \Omega, I, G)$ procedure (Algorithm 1). It holds that*

1. *If $state(s_0) = goal$ then T_s contains a solution tree for $\mathcal{P}$ and $\mathcal{P}$ has a solution.*
2. *If $\mathcal{P}$ has a solution then eventually s_0 becomes a goal node.*
3. *$Plan(F, A, \Omega, I, G)$ returns $NULL$ iff $\mathcal{P}$ does not have a solution.*

Proof. Before the **while** loop of Algorithm 1, it is easy to see that the state of s_0 is set correctly (Lines 3-7). Hence, the state of every node in the search graph T_s before the **while** loop is set correctly as T_s contains only one node s_0. Inside **while** loop, the state of nodes is changed only by the $explore$ procedure (Line 14). Looking into Algorithm 2 and due to Propositions 6 and 7, one can see that the state of every node in the search graph is always set correctly after the execution of $explore(s)$ (Line 14) in each iteration and so is it before the execution of $explore(s)$.

1. Since the state of every node in the search graph is set correctly, this is obvious due to Theorems 6 and 1.
2. One can see that after the exploration of an unexplored node s, $explore(s)$ created and added to the T_s every possible child and every possible outgoing edge from s. By Theorem 7 and due to the correct state of nodes in the graph, s_0 cannot be a dead node during the search. This means that the search will not terminate until s_0 becomes goal or T_s has been fully expanded and becomes complete (there are no more unexplored nodes in the search graph). In the first case clearly we have the proof. In the second case, T_s is the complete search graph for $\mathcal{P}$ and, by Theorem 1, it contains a solution tree for $\mathcal{P}$. This implies that s_0 is a goal node due to the correct state of nodes in T_s (proof).
3. $Plan(F, A, \Omega, I, G)$ returns $NULL$ iff either s_0 is dead or T_s is complete and s_0 is not a goal node iff $\mathcal{P}$ does not have a solution (Theorem 7) or T_s is complete and T_s does not contain a solution tree for $\mathcal{P}$ iff $\mathcal{P}$ does not have a solution (Theorem 1) (proof).

□

5. PrAO: A New AND/OR Search Algorithm for Contingent Planning

This section describes a new AND/OR search algorithm, called PrAO (Pruning AND/OR Search), for contingent planning. PrAO is similar to the standard AND/OR graph search algorithm presented previously 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 nodes and edges of a potential solution and eliminates useless parts of the graph. We start by introducing the notion of a variant search graph for more convenience to use with the minimal-DNF representation.

Definition 6. *Let $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$ be a contingent planning problem. A* search graph *is a labeled transition graph $\langle \mathcal{S}, \mathcal{T}, s_0\rangle$ 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 and $s_0 \in \mathcal{S}$; and*
- *$\mathcal{T}$ is a set of transitions, $\mathcal{T} \subseteq \mathcal{S} \times (A \cup \Omega) \times \mathcal{S}$, such that*
  - *for each $(s, \omega, s_1) \in \mathcal{T}$ such that $\omega \in \Omega$, $s \models pre(\omega)$ and there exists exactly one $(s, \omega, s_2) \in \mathcal{T}$ such that $\{s_1, s_2\} = \{s_\omega^+, s_\omega^-\}$; and*
  - *for each $(s, a, s_1) \in \mathcal{T}$ such that $a \in A$, $s \models pre(a)$ and $s_1 = \Phi_{DNF}(a, s)$.*

Observe that this graph is similar to the one used in the standard AND/OR search previously presented, except:

- Nodes in the graph are DNF-states instead of belief states, the start node s_0 is a DNF-state Δ_0 that represents the description of the initial state (I) as well as the initial belief state of the problem.
- The transitions in the graph are regulated by the transition function (for both actions and observations) for DNF-states instead of that for belief states.

Due to the equivalency between the transition functions for DNF-states and belief states (Theorems 2, 4, and 5), it is easy to see that all the theoretical results presented earlier in this paper for the standard AND/OR search graph are also true for this variant search graph. We will use the same notions for this variant search graph as for the standard search graph. Note that, although every solution tree in a standard search graph is a solution tree in the complete variant search graph for the same contingent planning problem, the converse is not necessarily true. The reason is that each belief state can be equivalent to multiple DNF-states while a DNF-state has only a unique extension (the belief state equivalent to the DNF-state) and, hence, multiple nodes of (though) different DNF-states in a variant search graph may represent the same belief state. Thus, let $T_g = \langle \mathcal{S}_g, \mathcal{T}_g, s_0 \rangle$ be a solution tree in a variant search graph for a problem $\mathcal{P}$ and if we replace each node in T_g by the belief state it represents, the obtained graph may not be a solution tree in the complete standard search graph for the same problem as it may contains a loop. However, one can prove (similarly to Theorem 1) that $tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ is a solution for $\mathcal{P}$ (note that $tree(\mathcal{S}_g, \mathcal{T}_g, s_0)$ is finite as T_g does not contain a loop in the variant search graph). Theoretically, since there is a finite number of DNF-states that are equivalent to each belief state, the complete variant AND/OR search graph for a contingent planning problem is still finite and hence, Theorems 1 and 8 (when we use this variant search graph in the standard search for contingent solutions) still hold for this variant search graph. As the matter of fact, we have never witnessed two different DNF-states representing a same belief state from the experiments on a large diverse set of problems.

Besides the use of DNF-states as nodes with the new function for transitions between them in a search graph[3], PrAO differs from the standard search algorithm previously presented due to pruning techniques incorporated in it. Those pruning techniques are built based on the propositions presented next. Let $\mathcal{P} = \langle F, A, \Omega, I, G \rangle$ be a contingent planning problem and $T_s = \langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ be a search graph constructed by PrAO during the search.

Proposition 9. *Let s be a node in T_s such that s is goal-reachable via a transition t, i.e., either $t \in A \wedge \exists (s, t, s') \in \mathcal{T} \wedge state(s') = goal$ or $t \in \Omega \wedge \exists (s, t, s_1), (s, t, s_2) \in \mathcal{T} \wedge state(s_1) = state(s_2) = goal$*

[3]For convenience, from now on we will call a variant search graph simply a search graph and a graph used in the standard search a standard search graph if the distinguishment is needed.

holds. Let $T'_s = \langle \mathcal{S}, \mathcal{T}', s_0 \rangle$ be the graph obtained by removing all the other transitions from s in $\mathcal{T}$, i.e., $\mathcal{T}' = \mathcal{T} \setminus \{(s, t_i, s_i) \in \mathcal{T} \mid t_i \neq t\}$. Let T_f and T'_f be the graphs obtained by fully expanding T_s and T'_s, respectively. If T_f contains a solution tree for $\mathcal{P}$ then so does T'_f.

Proof. Assume that T_f contains a solution tree T_g for $\mathcal{P}$. If s is not a node in T_g then, clearly, T_g is also a subgraph of T'_f as T'_f differs from T_f by only some transition(s) from s. Thus, T'_f contains a solution tree (T_g) for $\mathcal{P}$. If s belongs to T_g and there is the transition t from s in T_g then T_g is a subgraph of T'_f too. Now we need to consider the last case that s belongs to T_g but the transition t from s does not belong to T_g. Let t' be the transition from s in T_g. By Proposition 4 and the proof for it, there exists a subgraph $T_{t'}$ of T_g and another subgraph T_t of T_f that are solution trees for the problem $\mathcal{P}_s = \langle F, A, \Omega, s, G \rangle$, where the transition t' from s is in $T_{t'}$ and the transition t from s belongs to T_t. Observe that, T_t is also a subgraph of T'_f and if we replace $T_{t'}$ with T_t in T_g we will obtain another solution tree for $\mathcal{P}$. This new solution tree is a subgraph of T'_f (proof). □

Proposition 9 shows that when a node s becomes goal-reachable via a transition (or a pair of dual transitions), we can remove all the other transitions from s and the search still will find a solution if the problem has a solution.

Proposition 10. *Let s be a dead node in T_s. Then, the removal of every incoming transition to s (s', t, s) and its dual $(s', t, s")$ if $t \in \Omega$ from the search graph does not lose any solution tree that T_s or its expanded search graphs may contain.*

Proof. Since s is a dead node, s cannot be a node in a solution tree for the problem (Proposition 5). Hence, any incoming transition to s (s', t, s) and its dual $(s', t, s")$ if $t \in \Omega$ cannot belong to a solution tree. This implies the proposition. □

Proposition 10 allows for removing all the incoming edges to a dead node and the and-edges that are dual with some of them from the search graph.

Thus, during the search, PrAO will remove from the search graph such transitions as mentioned in Propositions 9 and 10. Consider the case that a transition s', t, s is removed from the search graph because s' is a goal node via another transition or $t \in \Omega$ and there exists $s', t, s"$ such that $s"$ is a dead node, s and thereby some of its descendants that can be reached from s may no longer be reached from the start node s_0. We say that a node s is *connected* (to the search graph) if s can be reached from the start node, i.e., there is a path (sequence of edges) in the graph leading from s_0 to s. Otherwise, s is said to be *isolated* or *disconnected* from the graph. It is easy to see that if a node is connected then so are all its descendants. The converse, however, does not always hold due to the fact that there can be multiple incoming transitions to a node.

Since every node in a solution tree can be reached from the start node, PrAO will consider only unexplored nodes that are connected to the search graph for expansion. However, an isolated node s can be connected to the search graph again if the exploration of a (connected) node s' creates a transition s', t, s and, in turn, the isolated descendants of s will be connected again if s is connected. For that reason, PrAO does not eliminate isolated nodes from the search graph. Instead, PrAO uses another boolean variable $connected(s)$ for each node s to keep track of its connection state for the selection of nodes to explore. Whenever an isolated node s is connected to the search graph again due to a newly created edge then PrAO changes its connection state to be connected. For a node s_i in the search graph, we say that the connection state of s_i is set correctly if $connected(s_i)$ is **true** if s_i is connected and it is **false** otherwise. We have similar definition for the state of a node.

Algorithm 6 $Plan(F, A, \Omega, I, G)$ The main procedure of the PrAO search for contingent planning

1: **Input**: A contingent problem $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$
2: **Output**: A solution for $\mathcal{P}$ if it has a solution or $NULL$ otherwise
3: Let s_0 be the initial belief state of $\mathcal{P}$ {The set of states satisfying I}
4: **if** $s_0 \models G$ **then**
5: **return** [] {Return the empty solution}
6: **else**
7: Set $state(s_0) = unexplored$, $connected(s_0) =$ **true**
8: **end if**
9: Let $\mathcal{S} = \{s_0\}, \mathcal{T} = \emptyset$ {Initialize the search graph with $(\{s_0\}, \emptyset, s_0)$}
10: **while true do**
11: **if** $\{s \in \mathcal{S} \mid state(s) = unexplored \wedge connected(s) =$ **true**$\} = \emptyset$ **then**
12: **return** $NULL$ {No solutions}
13: **end if**
14: Let s be the connected unexplored node with the best heuristic in $\mathcal{S}$
15: $explore(s)$ {Expand and update the search graph by exploring the node s}
16: **if** $state(s_0) = goal$ **then**
17: **return** $tree(\mathcal{S}, \mathcal{T}, s_0)$ {Return a solution for the problem $\mathcal{P}$}
18: **end if**
19: **if** $state(s_0) = dead$ **then**
20: **return** $NULL$ {No solutions}
21: **end if**
22: **end while**

The main procedure $Plan(F, A, \Omega, I, G)$ of PrAO is implemented by Algorithm 6. This procedure is similar to that for the standard search (Algorithm 1) with the following exceptions: Line 6 additionally sets s_0 to be connected and Lines 10 & 13 also consider the connected condition besides the unexplored condition of nodes. It is briefly described as follows. 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 initial DNF-state 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 set as unexplored and connected (Line 6) if s_0 does not satisfy the goal G. At each iteration, PrAO executes the following tasks:

- Select the connected 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 (Lines 10-13);
- Expand and update the graph accordingly by exploring the node s (Line 14); and
- Check the state of the root node s_0: if s_0 becomes a goal node then extract and return the solution (Lines 15-18). If s_0 is a dead node then terminate the search with no solution (Lines 19-21).

The $explore$ procedure is modified to incorporate the pruning techniques as shown in Algorithm 7 and its sub-procedures. Observe that the extraction of a solution tree is not needed in this procedure.

In this new version of $explore(s)$, one can see that Lines 5 & 18 preemptively apply Procedure 10 to not add incoming edges to dead nodes or their dual and-edges to the graph. The pruning techniques, that rely on Propositions 9 and 10, are also implemented in the $goal\_propagation$ and $dead\_propagation$ procedures to remove the redundant transitions and isolate (change the connection state of) the relevant nodes accordingly

Algorithm 7 $explore(s)$ Expanding the search graph by exploring a connected unexplored node s

1: **Input**: (connected unexplored) node $s \in \mathcal{S}$
2: **for** each action $a \in A$ **do**
3: **if** $s \models pre(a)$ **then**
4: Compute $s' = \Phi(a, s)$ {Compute a successor (or-child) of s}
5: **if not** (($s' \in \mathcal{S}$ **and** $state(s') = dead$) **or** ($s' = s$)) **then**
6: $expand(s, a, s')$ {Extend the graph with (only new) node s' and edge (s, a, s'), Algorithm 8}
7: **if** $state(s') = goal$ **then**
8: Set $state(s) = goal$
9: $goal\_propagation(s, a)$ {Algorithm 10}
10: **return**
11: **end if**
12: **end if**
13: **end if**
14: **end for**
15: **for** each observation $\omega \in \Omega$ **do**
16: **if** $s \models pre(\omega)$ **and** $s \not\models \ell(\omega)$ **and** $s \not\models \overline{\ell(\omega)}$ **then**
17: Compute $s_1 = s^+_\omega$, $s_2 = s^-_\omega$ {Compute a pair of dual and-children of s}
18: **if not** (($s_1 \in \mathcal{S}$ **and** $state(s_1) = dead$) **or** ($s_2 \in \mathcal{S}$ **and** $state(s_2) = dead$)) **then**
19: $expand(s, \omega, s_1)$ {Extend the graph with node s_1, if it is a new node, & and-edge (s, ω, s_1)}
20: $expand(s, \omega, s_2)$ {Extend the graph with node s_2, if it is a new node, & and-edge (s, ω, s_2)}
21: **if** $state(s_1) = state(s_2) = goal$ **then**
22: Set $state(s) = goal$
23: $goal\_propagation(s, \omega)$ {Algorithm 10}
24: **return**
25: **end if**
26: **end if**
27: **end if**
28: **end for**
29: **if** $\exists(s, t, s') \in \mathcal{T}$ **then**
30: Set $state(s) = explored$
31: **else**
32: Set $state(s) = dead$ {The explored node s is dead if there are no outgoing transitions from s}
33: $dead\_propagation(s)$ {Algorithm 12}
34: **end if**

when each goal or dead node is detected during the recursive execution of the procedures. We will detail this later after considering the (variant) $expand$ procedure.

The variant $expand(s)$ procedure, where s is a node being explored, is implemented in Algorithm 8. When a new edge from the node s to a successor s' of s is added to the graph (Line 2), if s' is an existing node in the graph and it was isolated by (the pruning techniques in) $goal\_propagation$ or $dead\_propsgation$ then it becomes connected again due to the newly added edge and so do the descendants of s'. The $reconnection\_propagation(s')$ (Line 4, Algorithm 9) recursively sets s' and its descendants as connected again if they were isolated. The correctness of the $reconnection\_propagation$ procedure

Algorithm 8 $expand(s, t, s')$ Set the state of s' and extend the graph with node s' (if s' is a new node) and edge (s, t, s')

1: **Input**: node s, $t \in A \cup \Omega$, node s'
2: $\mathcal{T} = \mathcal{T} \cup \{(s, t, s')\}$ {Extend the graph with the edge (s, t, s')}
3: **if** $s' \in \mathcal{S}$ **then**
4: $reconnection\_propagation(s')$ {Set s' and all descendants of s' as connected due to the new edge (s, t, s'), if they were isolated, Algorithm 9}
5: **else**
6: Set $connected(s') =$ **true**
7: **if** $s' \not\models G$ **then**
8: Set $state(s') = unexplored$
9: **else**
10: Set $state(s') = goal$
11: **end if**
12: $\mathcal{S} = \mathcal{S} \cup \{s'\}$ {Extend the graph with the new node s'}
13: **end if**

Algorithm 9 $reconnection\_propagation(s)$ Set s and all descendants of s (recursively) as connected, if they were isolated

1: **Input**: node s
2: **if** $connected(s) =$ **false then**
3: Set $connected(s) =$ **true**
4: **for** each $(s, t, s') \in \mathcal{T}$ **do**
5: $reconnection\_propagation(s')$
6: **end for**
7: **end if**

(Algorithm 9) is validated in the following proposition.

Proposition 11. *Let s be a node being explored, s' be a child of s due to a transition (s, t, s'). If the connection state of every node in the search graph is correctly set before adding the edge (s, t, s') to the graph (precondition) and s' is an existing node in the graph, then after adding (s, t, s') to the graph and executing $reconnection\_propagation(s')$ the connection state of every node in the search graph is correctly set.*

Proof. Observe that both the addition of edge (s, t, s') and the execution of $reconnection\_propagation(s')$ can affect only the connection state of node s' and its descendants but they do not affect the other nodes. Thus, the connection state of the nodes other than s' and its descendants remains correct. Thus, we only need to prove the proposition for s' and its descendants. We consider the following two cases.

- Case $connected(s') =$ **true**: the proof is trivial due to the precondition assumption and the fact that the execution of $reconnection\_propagation(s')$ does not really change anything in this case.
- Case $connected(s') =$ **false**: due to the precondition assumption, s is connected and hence so are s' and its descendant after adding the edge (s, t, s'). Thus, Line 3 correctly sets s' to be connected in the execution of $reconnection\_propagation$. This procedure will recursively set each descendant of s'

as connected to (Lines 4-6) and terminates only when all descendants of s' is set as connected or there is such a node s" such that $connected(s") =$ **true** holds before $reconnection\_propagation(s")$ may set it. Due to the precondition assumption, $connected(s") =$ **true** means that s" is (really) connected and so are its descendants. Thus the termination of the procedure at this node still assures that every descendant of s' is set connected.

In both cases, we showed that the connection state of s' and its descendants is set correctly after adding (s, t, s') to the graph and executing $reconnection\_propagation(s')$ (proof). □

Similarly, we have the following proposition for the $expand$ procedure (Algorithm 8).

Proposition 12. *Let s be a node being explored, s' be a child of s due to a transition (s, t, s'). If the connection state of every node in the search graph is set correctly before the execution of $expand(s, t, s')$ (precondition), then after executing $expand(s, t, s')$ the connection state of every node in the search graph is correctly set.*

Proof. If s' is an existing node in the graph then the proof is trivial due to Proposition 11. Otherwise, the connection state of all the nodes except s' does not change and it is easy to see that $expand(s, t, s')$ sets only the connection state of s' as connected (Line 6) in this case. The correctness of the setting connection state for all the nodes in the graph, hence, is obvious due to the precondition (proof). □

Algorithm 10 $goal\_propagation(s, t_g)$

1: **Input**: Node $s, t_g \in A \cup \Omega$
2: **for** each $(s, t, s') \in \mathcal{T}$ such that $t \neq t_g$ **do**
3: $\mathcal{T} = \mathcal{T} \setminus \{(s, t, s')\}$ {Remove all transitions from s except the new one that makes s become goal}
4: $isolation\_propagation(s')$ {Set s' and its descendants as isolated if there are no more edges connecting them to the graph after removing (s, t, s'), Algorithm 11}
5: **end for**
6: **for** each $(s_1, t, s) \in \mathcal{T}$ **do**
7: **if** $t \in A$ **then**
8: Set $state(s_1) = goal$
9: $goal\_propagation(s_1, t)$
10: **else**
11: Let s_1, t, s_2 be the dual and-edge of (s_1, t, s) in $\mathcal{T}$
12: **if** $state(s_2) = goal$ **then**
13: Set $state(s_1) = goal$
14: $goal\_propagation(s_1, t)$
15: **end if**
16: **end if**
17: **end for**

In the $goal\_propagation$ procedure (Algorithm 10) the pruning is implemented in Lines 2-5 aimed at removing every transition (s, t, s'), different from the (first) transition (or pair of dual transitions) that makes s goal-reachable, and setting the node s' and its descendants as isolated (Procedure $isolation\_propagation(s')$, Line 4) if there does not exist another transition that connects them to the graph after the removal of edge (s, t, s'). We have the following proposition for the correctness of the $isolation\_propagation$ (Algorithm 11)

Algorithm 11 $isolation\_propagation(s)$ Set s as isolated and recursively set the descendants of s, that cannot be reached from s_0 by a path not via s, as isolated

1: **Input**: node s
2: **if** $connected(s) =$ **true and** $\{(s_1, t, s) \in \mathcal{T} \mid connected(s_1) =$ **true**$\} = \emptyset$ **then**
3: Set $connected(s) =$ **false**
4: **for** each $(s, t, s_2) \in \mathcal{T}$ **do**
5: $isolation\_propagation(s_2)$
6: **end for**
7: **end if**

Proposition 13. *Let (s, t, s') be a transition in the search graph. If the connection state of every node in the search graph is set correctly (precondition), then after removing (s, t, s') from the graph and executing $isolation\_propagation(s')$ the connection state of every node in the search graph is set correctly.*

Proof. It is easy to see that the removal of (s, t, s') and the execution of $isolation\_propagation(s')$ may only change the connection state of s' and its descendants from being connected to be isolated and they do not affect the connection state of other nodes in the search graph. Hence, after removing (s, t, s') from the graph and executing $isolation\_propagation(s')$, the connection state of other nodes in the search graph is still set correctly due to the assumption of the precondition. Thus, we only need to consider the node s' and its descendants. Consider the following two cases about the condition in Line 2 at the beginning of the execution of $isolation\_propagation(s')$.

- Case the condition is not satisfied: then the procedure terminates immediately without changing anything. Due to the precondition, we have that '$connected(s') =$ **true**' holds and, hence, '$\{(s_1, t, s) \in \mathcal{T} \mid connected(s_1) =$ **true** $= \emptyset$' does not hold. This means that there is an incoming edge to s' from a connected node (after the removal of (s, t, s')). Hence, s' is still connected and so are its descendants. The connection state of s' and its descendants, hence, is obviously correctly set due to the precondition and the early termination of the procedure.

- Case the condition is satisfied: then there does not exist an incoming edge to s' from a connected node. Hence, the setting at Line 3 for s' to be isolated it correct. Lines 4-6 recursively execute the same procedure for every child of s', every child of each child of s', and so on. For any child or deeper descendant of s', either the procedure early terminates (at Line 2) or not, the connection state of the node and its descendants is always correctly set as proved for s'. Thus, after the completion of recursively executing $isolation\_propagation(s')$, the connection state of s' and all of its descendant is set correctly (proof).

The proposition has been proved in both cases. □

The correctness of the $goal\_propagation$ procedure (Algorithm 10) is shown in the following proposition.

Proposition 14. *Let s be a node being explored. Let t_g be either an action in A or an observation in Ω such that if $t_g \in A$ then there exists an or edge (s, t_g, s') in $\mathcal{T}$, where s' is a goal node and if $t_g \in \Omega$ then there exist a pair of dual and-edges (s, t_g, s_1) and (s, t_g, s_2) in $\mathcal{T}$, where both s_1 and s_2 are goal. We have the following.*

1. *If the state of every node in the graph but s before adding the transition(s) of t from s to the graph is set correctly, then after adding the transition(s) and executing $goal\_propagation(s, t_g)$ the state of every node in the graph including s is set correctly.*

2. *If the connection state of every node in the graph before executing $goal\_propagation(s, t_g)$ is set correctly, then after the execution of $goal\_propagation(s, t_g)$ the connection state of every node is set correctly.*

Proof. 1. Observe that this procedure changes the state of only goal nodes as $goal$ and it does not affect the state of other nodes. The proof is, hence, similar to that for Proposition 6.

2. This follows directly from Proposition 13

□

In the $dead\_propagation(s)$ procedure (Algorithm 12) the pruning is implemented in Lines 2-8. First, it removes each incoming edge (s_1, t, s) to a dead node s (Line 3) and if t is an observation then it also removes its dual and-edge (s_1, t, s_2) (Line 6) and, since s_2 and its descendants may become isolated due to the removal of (s_1, t, s_2), it then executes the $isolation\_propagation(s_2)$ (Line 7) to update the connection state of s_2 and its descendants. We have the following proposition for the correctness of the $dead\_propagation$ procedure.

Proposition 15. *Let s be a dead node in the search graph.*

1. *Suppose that the state of every node in the graph was set correctly before s becomes dead. Then after the execution of $dead\_propagation(s)$, the state of every node in the graph is correctly set.*

2. *If the connection state of every node in the graph before executing $dead\_propagation(s)$ is set correctly, then after the execution of $dead\_propagation(s)$ the connection state of every node is set correctly.*

Proof. 1. Observe that this procedure changes the state of only dead nodes as $dead$ and it does not affect the state of other nodes. The proof is, hence, similar to that for Proposition 7.

2. This follows directly from Proposition 13

□

Observe that, due to Proposition 15 and the execution of $dead\_propagation(s)$ for each dead node s in the graph, every dead node in the search graph is disconnected and completely isolated from all the other nodes. Hence, the $isolation\_propagation$ from an unexplored node cannot reach s to change the value of $connected(s)$ and thereby $connected(s)$ can be **true**. However, this does not affect the correctness of the search algorithm as PrAO never explores or creates a transition from or to a dead node. The purpose of the pruning techniques is to avoid exploring (unexplored) disconnected nodes and an unexplored node is disconnected because it does not have a connected ancestor, except the start node at the beginning of the search. Observe that an ancestor of an unexplored node cannot be a dead node or a goal node as a goal node can have only goal descendants due to the pruning so its state must be explored. Thus, the pruning techniques and $isolation\_propagation$ and $reconnection\_propagation$ actually can affect only nodes with unexplored or explored states.

Similar to Proposition 8, we have the following proposition for this search algorithm.

Algorithm 12 $dead\_propagation(s)$

Input: A dead node $s \in \mathcal{S}$
for each $(s_1, t, s) \in \mathcal{T}$ **do**
  $\mathcal{T} = \mathcal{T} \setminus \{(s_1, t, s)\}$ {Remove every incoming edge to a dead node s, Proposition 10}
  if $t \in \Omega$ **then**
    Let (s_1, t, s_2) be the dual and-edge of (s_1, t, s) from s_1 in $\mathcal{T}$ $\{s_2 \neq s\}$
    $\mathcal{T} = \mathcal{T} \setminus \{(s_1, t, s_2)\}$ {Remove also the dual and-edge of (s_1, t, s), Proposition 10}
    $isolation\_propagation(s_2)$ {Set s_2 and its descendants as isolated if they are no longer connected due to the removal of the edge (s_1, t, s_2)}
  end if
  if $\nexists (s_1, t', s') \in \mathcal{T}$ **then**
    Set $state(s_1) = dead$ {This is due to the removal of (s_1, t, s) & (s_1, t, s_2) if $t \in \Omega$}
    $dead\_propagation(s_1)$
  end if
end for

Proposition 16. *If a node ever becomes a goal node or a dead node in the search graph then its state can never change thereafter.*

Proof. The proof is rather easy and similar to that for Proposition 8. □

Due to the existence of disconnected nodes in this search, we relax the notion of solution tree as follows.

Definition 7. *A search graph $T_s = \langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ for a contingent planning problem $\mathcal{P} = \langle F, A, \Omega, I, G \rangle$ is called a* extended solution tree for $\mathcal{P}$ *if:*

1. *Every* connected *leaf node in $\mathcal{S}$ satisfies the goal G,*
2. *From each* connected *non-leaf node $s \in \mathcal{S}$ there exists either exactly one outgoing or-edge $(s, a, s') \in \mathcal{T}$ and $s' \in \mathcal{S}$ or exactly a pair of outgoing dual and-edges $(s, \omega, s_1), (s, \omega, s_2) \in \mathcal{T}$ and $s_1, s_2 \in \mathcal{S}$, and*
3. *The graph does not contain a loop, i.e., $\mathcal{T}$ does not contain a set of edges of the form $\{(s_1, t_1, s_2), (s_2, t_2, s_3), \ldots, (s_n, t_n, s_1)\}$, where $n \geq 1$ and $s_1, s_2, \ldots, s_n$ are connected.*

Observe that a solution tree is also an extended solution tree for the same problem. The converse is not necessarily true since there may exist some leaf node s in an extended solution tree such that s does not satisfy the goal if s is not connected. We have the following theorem.

Theorem 9. *Let $T_s = \langle \mathcal{S}, \mathcal{T}, s_0 \rangle$ be an AND/OR search graph for a contingent planning problem $\mathcal{P}$. If T_s is an extended solution tree for $\mathcal{P}$, then $tree(\mathcal{S}, \mathcal{T}, s_0)$ is a solution for $\mathcal{P}$.*

Proof. Observe that, by definition, $tree(\mathcal{S}, \mathcal{T}, s_0)$ is constructed from the start node s_0 and based on only connected nodes, i.e., nodes that can be reached from s_0. Thus the isolated nodes in the graph do not affect $tree(\mathcal{S}, \mathcal{T}, s_0)$. This means that if we remove all the isolated node from the graph to obtain $T_g = \langle \mathcal{S}_g, \mathcal{T}, s_0 \rangle$ then $tree(\mathcal{S}_g, \mathcal{T}, s_0) = tree(\mathcal{S}, \mathcal{T}, s_0)$. By definition, T_g is a solution tree for $\mathcal{P}$. Thus, $tree(\mathcal{S}, \mathcal{T}, s_0)$ is a solution tree for $\mathcal{P}$ (proof)[4]. □

[4]This theorem can be also proved exactly the same as that for the first statement of Theorem 1.

Let T_s be an extended solution for a contingent planning problem $\mathcal{P}$. Informally, Theorem 9 and the proof for it show that if we ignore or remove all isolated nodes in the extended solution tree T_s, then T_s is also a solution tree for $\mathcal{P}$ and a solution for $\mathcal{P}$ can be achieved by extracting from this extended solution tree in the same manner as if it was a solution tree for $\mathcal{P}$.

Lemma 3. *Let $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$ be a contingent planning problem, $T_s = \langle \mathcal{S}, \mathcal{T}, s_0\rangle$ be a search graph constructed by the $Plan(F, A, \Omega, I, G)$ procedure (Algorithm 6). Let T_c be the complete search graph constructed by the $Plan(F, A, \Omega, I, G)$ without applying the pruning techniques. Then every goal node in T_c is not a dead node in T_s.*

Proof. By *goal distance of s* we call the maximum length (number of transitions) of a loop-free path from s to a leaf node that satisfies the goal G. We will prove this lemma by induction on the goal distance of a goal node s as follows.

- Base case the goal distance of a goal node s is 0: this means that $s \models G$. It it easy to see that s is never be explored so it cannot become a dead node (by definition, a dead node must be a node that has been explored, proof).
- Inductive step: suppose that the lemma is true for every goal node with a goal distance less than k, for some $k \geq 1$. We will prove that it is true for every goal node s with a goal distance k. Assume the contrary, i.e., there exists such a node s that is a goal node in T_c with the goal distance k and s is a dead node in T_s. This means that s has been explored and every child of s became dead (s has at least a (goal) child as the goal distance of s is k, $k \geq 1$, in T_c). This implies that the goal child of s in T_c was a dead node, a contradiction with the conjunction of the inductive hypothesis and Proposition 16. We have the proof.

□

We have the following theorem for the PrAO search algorithm.

Theorem 10. *Let $\mathcal{P} = \langle F, A, \Omega, I, G\rangle$ be a contingent planning problem and $T_s = \langle \mathcal{S}, \mathcal{T}, s_0\rangle$ be a search graph constructed by the $Plan(F, A, \Omega, I, G)$ procedure (Algorithm 6). It holds that*

1. *If $state(s_0) = goal$ then T_s is an extended solution tree for $\mathcal{P}$ and $\mathcal{P}$ has a solution.*
2. *If $\mathcal{P}$ has a solution then eventually s_0 becomes a goal node.*
3. *$Plan(F, A, \Omega, I, G)$ returns $NULL$ iff $\mathcal{P}$ does not have a solution.*

Proof. Similarly to Theorem 8 and based on Propositions 14 and 15, one can prove that the state of every node in the search graph is always set correctly after the execution of $explore(s)$ (Line 14) in each iteration of the **while** loop in Algorithm 6 and so is it before the execution of $explore(s)$.

1. First we will prove that every connected node in T_s is a goal node. Indeed, due to the pruning performed by $goal\_propagation$ for all the goal nodes, each goal node has exactly either one goal or-child or one pair of goal and-children, if it is not a leaf node (the second property of a (extended) solution tree is satisfied). This implies that each goal node can have only goal descendants and cannot have a descendant that is not a goal node. Since $state(s_0) = goal$, s_0 is a goal node and, hence, it can have only goal descendants. This means that every connected node in T_s is a goal node. Thus,

every connected leaf node in T_s satisfy the goal G (the first property of an extended solution tree is satisfied). Now we need to prove that T_s does not contain a loop within connected nodes. Assume the contrary, that $\mathcal{T}$ contains a set of edges as follows:

$\{(s_1, t_1, s_2), (s_2, t_2, s_3), \ldots, (s_n, t_n, s_1)\}$, where $n \geq 1$ and $s_1, s_2, \ldots, s_n$ are connected.

Due to the conditions in Lines 5 & 16 of Algorithm 7, there cannot be a transition of the form (s, t, s) in the graph. Hence, $n > 1$. Since $s_1, s_2, \ldots, s_n$ are connected, they are all goal nodes as proved previously. Let s_i be the node that became goal first among the n nodes, $1 \leq i \leq n$. This means that at the time s_i became goal, s_{i+1} was not a goal node, where s_{n+1} denotes s_1. We consider two cases. $s_i \models G$, then s_i would have never been explored and, hence, there can not exist the edge from s_i, t_i, s_{i+1}, a contradiction. $s_i \not\models G$, then s_i became goal due to an or-edge or a pair of dual and-edges from s_i to goal node(s). If s_i, t_i, s_{i+1} belong to the edge(s), then s_{i+1} became goal before s_i, a contradiction. Otherwise, s_i, t_i, s_{i+1} would be removed by $goal\_propagation(s_i, t_g)$ and s_i, t_i, s_{i+1} would not be created again by the search, since s_i was a goal node and it would not be explored again. This leads to another contradiction. Thus, $\mathcal{T}$ does not contain such a loop. Thus, all the three properties of an extended solution tree are satisfied by T_s, i.e., T_s is an extended solution tree. By Theorem 9, $tree(\mathcal{S}, \mathcal{T}, s_0)$ is a solution for $\mathcal{P}$ (proof).

2. Let T_c be the complete graph obtained after it has been fully expanded by the search *without* applying the pruning techniques. Similar to the proof for the second statement of Theorem 1, one can prove that T_c contains a solution tree for $\mathcal{P}$. Due to the fact that the number of all possible belief states is finite and there is a finite number of DNF-states equivalent to each belief state, T_c is finite. Similar to the proof for the second statement of Theorem 8, the search will eventually terminate and s_0 becomes a goal node.

   Now suppose the search removes all the edges from goal nodes and the edges to dead nodes as presented earlier but it explores every unexplored nodes including the isolated nodes. By Propositions 9 and 10, the complete graph T_{ch} when it is fully expanded still contains a solution tree so eventually s_0 becomes a goal node.

   Finally, we consider the search that fully applies the pruning techniques (PrAO), i.e., it only explores connected unexplored nodes. Again, we consider the complete graph T_{cp} obtained after it is fully expanded (by PrAO). We prove that the complete graph T_{cp} in this case also contains a solution tree for $\mathcal{P}$. Assume the contrary, i.e., T_{cp} does not contain a solution tree for $\mathcal{P}$. The only possible reason for this because PrAO never explores some isolated nodes to make them become goal nodes and part of a solution tree and if each of these node is not explored, i.e., not a goal node later, then T_{cp} does not contain a solution tree for $\mathcal{P}$. Let s be such a node then s is a goal node in a solution tree T_g that is a subgraph of T_{ch}. Since T_g and T_{cp} share the common start node s_0, which is connected in T_{cp} and an ancestor of s, let s_1 be the closest ancestor of s such that s_1 is connected in T_{cp}, there exists a transition s_1, t, s_2 in T_g such that s_2 is isolated in T_{cp}, i.e., s_1, t, s_2 is not in T_{cp}, and s_2 is either an ancestor of s or $s_2 = s$. Since s is a node that must be explored, s_1 is not a goal node (otherwise, exploring s is not necessary in the search by PrAO as an ancestor s_1 of it can be goal-reachable without exploring it). Since s_1 is in T_{cp} and connected in T_{cp} and $s_1 \not\models G$ (otherwise there does not exist the transition s_1, t, s_2), s_1 must be explored in T_{cp} as T_{cp} has been fully expanded by PrAO. This implies that s_1, t, s_2 was created by PrAO and it is not in T_{cp}. This means that s_1, t, s_2 was removed by either $goal\_propagation(s_1)$ or $dead\_propagation(s_2)$ (by PrAO). As s_1 is not a goal node, this implies that s_2 is a dead node in T_{cp}. By Lemma 3, this is impossible as s_2 is a goal node in T_{ch} and, hence,

it is also a goal node in T_c.

3. We prove that if $Plan(F, A, \Omega, I, G)$ returns $NULL$ then $\mathcal{P}$ does not have a solution. Assume the contrary, then s_0 cannot be a dead node (Lemma 3) and will become goal (by the second statement of this theorem). This implies that $Plan(F, A, \Omega, I, G)$ will return a solution, a contradiction.

   Now we prove that if $\mathcal{P}$ does not have a solution then $Plan(F, A, \Omega, I, G)$ returns $NULL$. Since $\mathcal{P}$ does not have a solution, s_0 cannot become goal (otherwise the problem has a solution due to the first statement of this theorem). Thus, $Plan(F, A, \Omega, I, G)$ terminates only when the start node s_0 is dead, i.e., $\mathcal{P}$ does not have a solution (Lemma 3), or the search graph is fully expanded and s_0 is not goal, i.e., $\mathcal{P}$ does not have a solution either due to the second statement of this theorem.

□

Theorem 10 shows that PrAO is sound and complete.
To illustrate the PrAO algorithm, we consider the following example.

Example 3. *Let $\mathcal{P} = \langle\{f, g, h\}, \{a, b, c, d, e, t, p_1, p_2\}, \{s_g\}, true, \{f, \neg g, h\}\rangle$ be a contingent problem. The specification of the actions of $\mathcal{P}$ are as follows.*

- $pre(a) = \emptyset$, $O(a) = \{\emptyset \rightarrow f\}$; [5]
- $pre(b) = \{f\}$, $O(b) = \{\emptyset \rightarrow \{\neg f, g\}\}$;
- $pre(c) = \{f\}$, $O(c) = \{\emptyset \rightarrow \{f, \neg g\}\}$;
- $pre(d) = \{\neg f, g\}$, $O(d) = \{\emptyset \rightarrow \{f, \neg g\}\}$;
- $pre(e) = \{g\}$, $O(e) = \{\emptyset \rightarrow \{\neg f, \neg g\}\}$;
- $pre(t) = \{\neg g\}$, $O(t) = \{\emptyset \rightarrow \{\neg f, \neg g\}\}$;
- $pre(p_1) = \{f, \neg g\}$, $O(p_1) = \{\emptyset \rightarrow h\}$;
- $pre(p_2) = \{\neg f, \neg g\}$, $O(p_2) = \{\emptyset \rightarrow \{f, h\}\}$;
- $pre(s_g) = \emptyset$ *and* $l(s_g) = g$.

Figure 2 shows a search graph for the problem $\mathcal{P}$, where the pruning is not applied, i.e., all the outgoing edges from every node, that has been explored, remain in the graph. In the figure, solid links represent or-edges and dash links represent and-edges. Two dual and-edges are connected by an arc. It is easy to see that the problem has different solutions, including the following transition trees:

1. $a \circ b \circ d \circ p_1$;

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

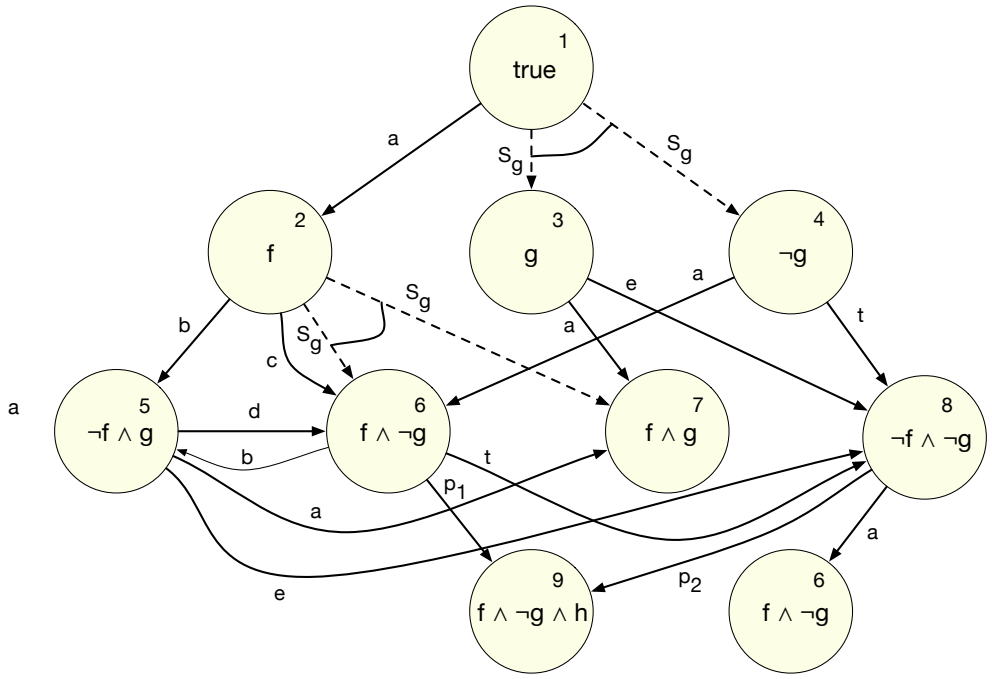

Figure 2: A search graph (without pruning) for the problem $\mathcal{P}$

2. $a \circ c \circ p_1$*; or*

3. $s_g(e \circ p_2 \mid t \circ p_2)$.

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. Initially, node 1 is created. The expansion of node 1 leads to the creation of nodes 2, 3, and 4 (and the addition of these nodes 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 and, hence, no isolation propagation is executed either. Since no successor node of node 1 exists in $\mathcal{S}$, no reconnection propagation is performed. We consider two orders of expansion by PrAO:

- *Assume that we expand nodes 2, 5, and 6 in this order and each expansion is executed in the order the actions are given. Expanding 2 results in the creation of nodes 5, 6, and 7 with the corresponding edges. No propagation is executed.*
  Expanding node 5 creates the edges $(5, a, 7)$, $(5, d, 6)$ to the already existing nodes 7 and 6 and the pair of a new node 8 and a new edge $(5, e, 8)$, denoted by (8, $(5, e, 8)$). The reconnection propagations at nodes 6 and 7 do not change anything since they are not isolated.
  Expanding nodes 6 creates the edges $(6, b, 5)$, $(6, t, 8)$, and the pair (9, $(6, p_1, 9)$). Since node 9 satisfies the goal G, 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 $(6, b, 5)$, $(6, t, 8)$. Nodes 5 and 8 still have another incoming edge so they are not isolated by isolation-propagation at this point. If node 5 is considered before node 2 then node 5 becomes goal first, the edge $(5, d, 6)$ is retained and the edges $(5, a, 7)$ and $(5, e, 8)$ are removed. Node 8 is isolated by the isolation-propagation as it does not have an incoming edge. Then

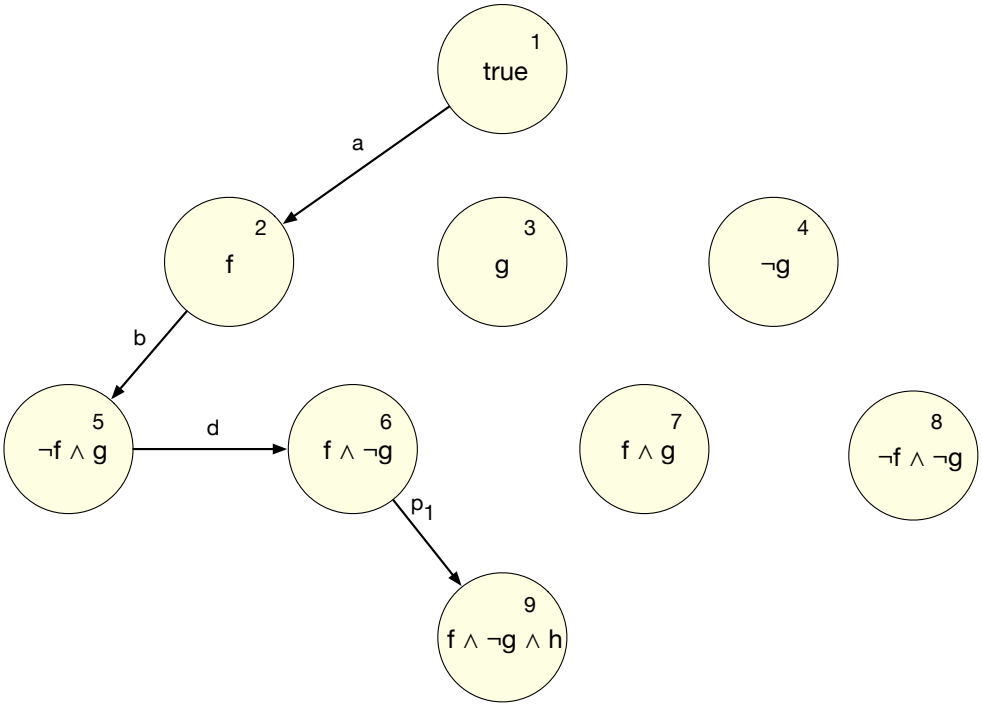

Figure 3: First solution: $a \circ b \circ d \circ p_1$

$goal\_propagation(2, b)$ removes every outgoing edges from node 2 but $(2, b, 5)$. Node 7 now is isolated. Then node 1 becomes goal and nodes 3 and 4 are isolated with no incoming edges. The first solution is found with the remaining graph illustrated in Figure 3.
Similarly, if node 2 is considered before node 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 node 3 and node 8. The expansion of node 3 creates nodes 7 and 8. Expanding node 8 creates nodes 6 and 9 and edges from node 8 to them. Node 9 becomes goal and, hence, so does node 8. The goal propagation removes $(8, a, 6)$, isolates 6, set node 3 as goal, removes $(3, a, 7)$, and isolates 7. Suppose that node 4 is better than node 2 and worse than node 6 and node 7, in terms of heuristic. Then PrAO selects node 4 to expand next, since nodes 6 and 7 are isolated. Expanding node 4 creates two edges from node 4 to nodes 6 and 8, reconnecting node 6 and setting node 4 as a goal node. The goal propagation results in the graph 5 and the third solution is achieved.*

6. The Contingent Planner DNF$_{ct}$ and Empirical Evaluation

6.1. The DNF$_{ct}$ *Planner*

DNF$_{ct}$ is a progression-based contingent planner that employs the PrAO search in belief space under the minimal-DNF representation extended in this paper for contingent planning solutions. DNF$_{ct}$ is built on top of the conformant planner DNF [18]. In this paper, DNF$_{ct}$ uses the heuristic function defined by the pair $\langle h_{goal}(\Delta), h_{lit}(\Delta)\rangle$ in the lexicographical order, where

- $h_{goal}(\Delta)$: the number of subgoals satisfied by the DNF-state Δ.

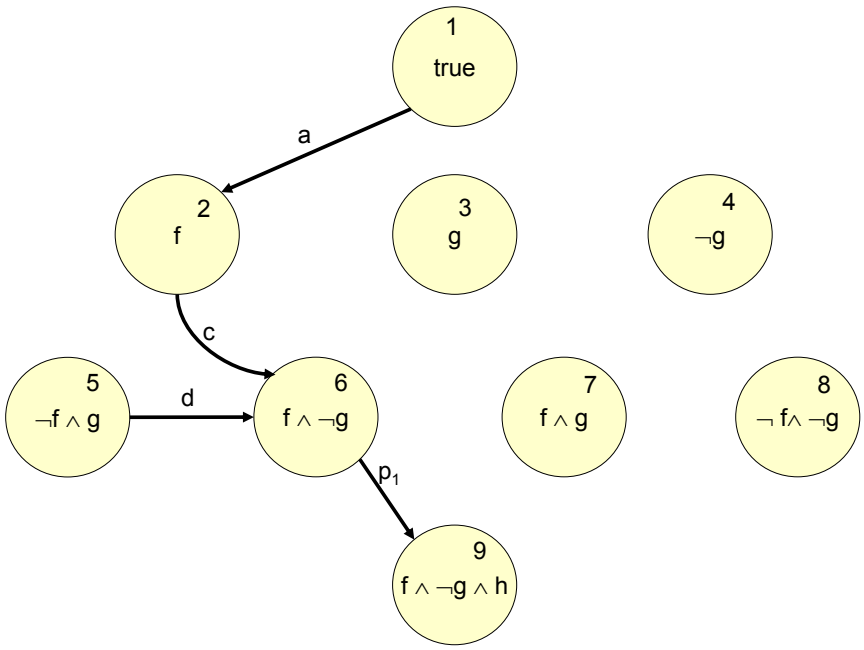

Figure 4: Second solution: $a \circ c \circ p_1$

- $h_{lit}(\Delta)$: the number of known literals in Δ.

The first component of this heuristic function is the same as that for DNF [18]. The second component—the number of known literals in the DNF-state—prioritizes expansion of nodes that contain less uncertainty and have a smaller size, as the more known literals in the DNF-state the less partial states the DNF-state contains in general. Note that this heuristic function differs than that used by DNF$_{ct}$ in the preliminary work [18]. As a consequence, one may observe a slight difference between the results DNF$_{ct}$ performed reported in the two papers. This work has tried with the heuristic function similar to that proposed by Hoffmann and Brafman [5, 10] but the performance did not improve. This is due to the incomplete information—that makes the estimated distance to the goal even less accurately, besides the relaxation of the problem.

6.2. Other Planners and Experimental Setup

This work compares DNF$_{ct}$ with CLG [1], contingent-FF [10], and POND 2.2 [7] obtained from their websites. These planners are known to be among the best currently available contingent planners. In the experiments, contingent-FF was executed with both options—i.e., with and without helpful actions—and only the best result for each instance is reported in this paper. POND was executed using the AO* search algorithm (aostar). As observed from the experiments, the execution time of CLG can vary in a very wide range (e.g., from a few to hundreds of seconds) for a same instance. Hence, for CLG, the best result of several execution times for each instance is reported.

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.

In the result tables, time, size, and depth denote the overall execution time, the number of actions in the solution, and the longest path from the start node to a leaf node of the solution tree, respectively. The results

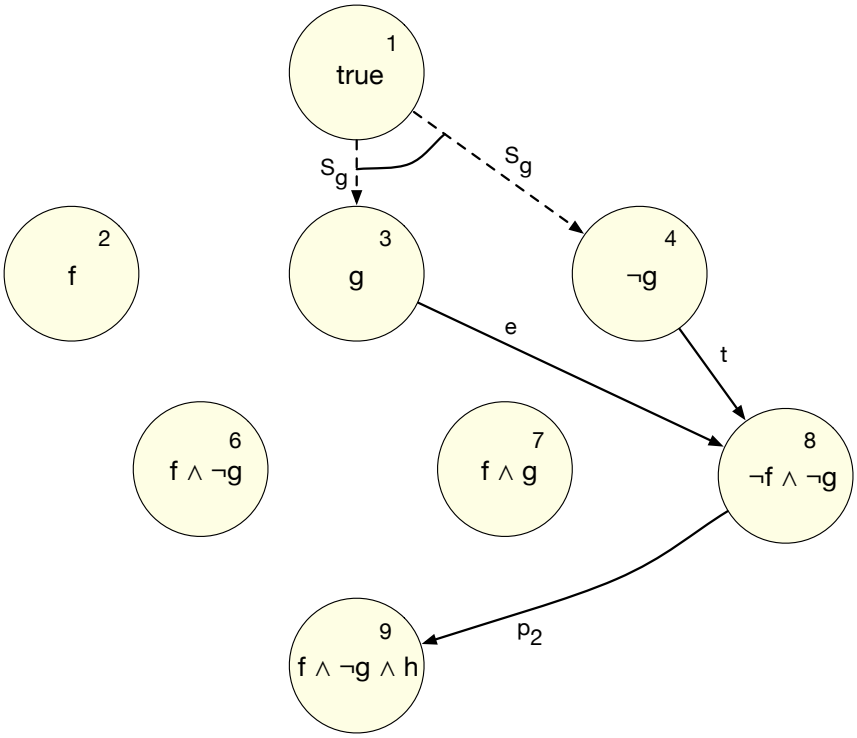

Figure 5: Third solution: $s_g(e \circ p_2 \mid t \circ p_2)$

of POND omit the depth of each solution, since POND does not report this value. Usually, the depth and the size of a solution tree are criteria for evaluation of the quality of the solution[6].

In the result 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).

Observe that even though DNF$_{ct}$ employs a greedy best first search in AND/OR graph for solutions, not an AO* search as contingent-FF and POND use[7], the quality of the solutions found by DNF$_{ct}$ is not worse than those found by the other planners, in general. I 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 can be very inaccurate in the presence of incomplete information.

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

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

[7]POND supports several options for search algorithms, this work selected AO* search option in the experiments.

6.3. Benchmarks

The benchmarks used in the experiments for this paper come from:

1. the distributions of contingent-FF and POND, including $btcs$, $btnd$, bts, $ebtcs$, and $ebtnd$—which are different variants of the bomb in the toilet—and the variants of $block$, $logistic$, $grid$, $medpks$, and $unix$ (Table 1);

2. the distribution of CLG, including $cballs$-n-m, $doors$-n, $localize$-n, and $wumpus$-n—which are variants of the respective grid domains (Table 2);

3. a modification of several challenging conformant domains, including $edispose$, $e1d$, ecc, and $epush$ (Table 3).

The contingent problems in the last set are variants of the following conformant domains in the grid family: $dispose$, 1-$dispose$, $corner$-$cube$, and $push$, respectively. These problems are modified by moving out some e-conditions of actions as the preconditions of the actions in the new problems and adding the corresponding sensing actions. This enforces the planners to perform sensing actions in order to obtain solutions for the problems. 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. Thus, without sensing, the agent cannot perform the actions and, hence, a solution for the problem cannot be achievable. The modifications in the domains $e1d$, ecc, and $epush$ have done similarly.

6.4. Results on Problems from contingent-FF and POND Distributions

Table 1 reports the experimental results on the problems from contingent-FF and POND distributions. As one can see, DNF$_{ct}$ outperforms all the other planners on seven out of twelve domains, including $btcs$, $btnd$, bts, $ebtcs$, $ebtnd$, $medpks$, and $unix$. DNF$_{ct}$ also scales up very well on these domains as it can solve the largest instances within a small total run-time, while the other planners cannot solve or spent much longer time for a solution for those instances. The sizes of 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 and preprocessing phase, e.g., for $unix$-2 DNF$_{ct}$ spent only 0.025 second for the search while it spent 0.62 second for the translation and preprocessing phase. In this group, DNF$_{ct}$ does not perform well on $block$, $clogistic$, $elogistic$, and $grid$. This is because the heuristic function mostly based on the number of satisfied subgoals, that DNF$_{ct}$ uses, is misleading on these 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 $block$, $clogistics$, and $elogistics$, and performs second best on most of other domains, except $btnd$ and $ebtnd$.

Problem	DNFct			CLG			contingent-FF			Pond	
	Time	Size	Depth	Time	Size	Depth	Time	Size	Depth	Time	Size
block-3	0.48	5	4	0.09	6	4	0.02	6	4	**0.01**	5
block-7	11.6	69	28	4.88	55	9	**0.46**	49	12	OM	
block-11		OM		**36.27**	115	18		TO		-	
btcs-70	**2.76**	139	70	14.61	140	140	123.64	139	70	74.04	139
btcs-90	**5.63**	179	90	41.35	180	180	476.8	179	90	TO	
btcs-150	**27.5**	**299**	**150**	379	300	300		MC		-	
btnd-70	**1.47**	**209**	72		NA		536.6	140	72	TO	
btnd-90	**2.28**	369	92		-		2070	**180**	92	-	
btnd-150	**7.47**	**449**	**152**		-			TO		-	
bts-70	**1.53**	139	70	11.69	70	70	1672	70	70	TO	
bts-90	**2.6**	179	90	29.83	90	90		TO		-	
bts-150	**9.73**	299	150	251	**150**	150		-		-	
clogistics-7	0.77	428	136	**0.16**	**210**	**22**		E		9.76	193
clogistics-L		OM		**148**	**37718**	**73**		-		OM	
ebtcs-70	**1.04**	139	70	25.49	209	71	63	139	70	24.69	139
ebtcs-90	**1.56**	179	90	71.19	269	91	255.5	179	90	TO	
ebtcs-150	**4.57**	**299**	150	658	449	150		MC		-	
ebtnd-70	**1.28**	276	72		NA		16.3	**208**	72	TO	
ebtnd-90	**1.91**	356	92		-		53.15	**268**	92	-	
ebtnd-150	**5.95**	**596**	**152**		NA			MC		-	
egrid-3	1.78	350	47	**0.75**	111	**28**	943	58	41	**105**	148
egrid-4	**3.06**	**849**	54	4.75	884	**48**		TO		OM	
egrid-5	10.1	1469	136	**1.65**	**208**	**40**		-		-	
elogistics-5	0.78	301	179	0.1	147	**21**	**0.02**	156	23	0.67	**143**
elogistics-7	0.9	428	136	0.14	**210**	**22**	**0.04**	223	23	0.95	212
elogistics-L		OM		**90.3**	**36152**	**73**		TO		OM	
grid-3	1.74	381	67	0.74	114	30	**0.06**	**23**	**23**	104	178
grid-4	2.9	849	63	4.62	872	51	**0.14**	**49**	**49**	OM	
grid-5	12.34	1337	81	1.55	212	**40**	**0.15**	**46**	46	-	
medpks-70	**1.49**	141	72	8.12	141	71	968.6	140	71	TO	
medpks-99	**4.57**	199	101	25.6	199	101		TO		-	
medpks-150	**9.93**	599	152	106.4	**299**	**151**		-		-	
unix-2	0.65	48	37	0.41	50	39	**0.13**	48	37	1.71	48
unix-3	**1.87**	111	84	5.39	113	86	3.84	111	84	OM	
unix-4	**16.1**	238	179	79.8	240	181	142.8	238	179	-	

Table 1: The performance of DNF$_{ct}$ and other planners on problems from contingent-FF and POND distributions

6.5. Results on Challenging Problems from CLG Distribution and Modification of Conformant Problems

Tables 2 and 3 contains the experimental results on the two set of challenging problems proposed by the authors of CLG and by this work, respectively.

Problem	DNFct			CLG			contingent-FF			Pond	
	Time	Size	Depth	Time	Size	Depth	Time	Size	Depth	Time	Size
cball-3-2	**0.86**	609	40	2.71	2641	34		TO		2.2	597
cball-3-3	**10.5**	8030	57	49.1	60924	48		-		39.7	4808
cball-5-1	**1.09**	119	68	14.7	586	65		-		527	199
cball-5-2	**21.5**	5165	107	289	72817	107		-		OM	
cball-8-1	**10.42**	307	162	540	2411	171		-		-	
cball-8-2	**780**	27k	281		TO			-		-	
cball-8-3		OM			-			-		-	
doors-7	**4.92**	2193	53	7.93	2153	51		E		17.99	2159
doors-9	**44.3**	45k	89	594	46024	95		-		1262	44082
doors-11		OM			TO			-		TO	
localize-5	**0.54**	48	31	0.6	112	24	42	53	53	TO	
localize-7	**0.69**	80	48	3.07	231	37		MC		-	
localize-9	**0.82**	110	61	13.49	386	50		-		-	
localize-11	**1.33**	177	90	49.9	577	63		-		-	
localize-13	**2.31**	216	136		OM			-		-	
wumpus-5	2	1227	35	**0.58**	754	41		E		4.65	587
wumpus-7	56.4	30319	69	**8.65**	6552	57		-		TO	
wumpus-10		OM		**1370**	280k	100		-		-	

Table 2: The results on challenging problems from CLG distribution

Most of these problems are harder than those reported in Table 1 due to the higher uncertainty in the initial belief state and, more importantly, in the e-conditions of the actions in the problems. contingent-FF and POND can only solve a few small instances, except those of the ecc domain on which contingent-FF scales up better than CLG and POND. Again, CLG is the only competitor of DNF$_{ct}$ on most problems in these two sets. 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 performs best in $wumpus$-n. DNF$_{ct}$ also offers the best performance on the modifications of conformant problems: $edis$-n-m, $e1d$-n-m, ecc-n-m and $epush$-n-m. Observe that, CLG scales up well on the first dimension (n) of $cball$-n-m, $edis$-n-m, $e1d$-n-m, and $epush$-n-m but it has trouble with the second dimension (m), on which DNF$_{ct}$ scales much better. In these problems, n denotes the size of the grid of $n \times n$ cells (locations) and m is the number of objects given in the problem. For example, in the instance $epush$-2-3, there are $2 \times 2 = 4$ cells in the grid and 3 objects given in this problem instance. In these problems, predicates of the form $at(p_{i,j})$ indicate that the robot is at the cell (i, j) of the grid (location $p_{i,j}$) and predicates of the form $obj\_at(o_k, p_{i,j})$ indicate that the object o_k is at the location $p_{i,j}$, for $i, j \leq n$ and $k \leq m$. Initially, the location of each object is unknown among the $n \times n$ locations. Hence, in the description of the initial world there is a set of m one-of

Problem	DNFct			CLG			contingent-FF			Pond	
	Time	Size	Depth	Time	Size	Depth	Time	Size	Depth	Time	Size
edisp-3-2	**0.64**	397	36	0.59	752	39		E		TO	
edisp-3-3	**1.58**	3891	57	6.45	8552	52		-		-	
edisp-5-1	**0.99**	98	60	2.85	177	60		-		-	
edisp-5-2	**3.45**	2753	97	31.3	7993	87		-		-	
edisp-10-1	**46.9**	400	233	140	1051	237		-		-	
edisp-10-2	**298.3**	31357	402		TO			-		-	
edisp-10-3		OM			-			-		-	
e1d-3-1	**1.15**	33	20	19.7	50	20		E		TO	
e1d-3-3	**1.74**	3557	88	392	15294	62		TO		-	
e1d-5-1	**1.57**	99	59	2061	200	58		-		-	
e1d-5-3	**88.5**	76088	417		TO			-		-	
e1d-10-1	**164**	399	233		-			-		-	
e1d-10-2	**461**	47652	665		-			-		-	
e1d-10-3		OM			-			-		-	
ecc-40-20	**1.15**	466	83	2089	275	75	37.1	288	63	TO	
ecc-75-37	**3.24**	903	204		TO		999	529	114	-	
ecc-99-49	**6.11**	1.1k	184		-		5.3k	697	150	-	
ecc-119-59	**9.18**	1.4k	292		-			TO		-	
epush-3-1	0.53	39	29	**0.23**	50	24	0.6	61	37	TO	
epush-3-3	**1.32**	1249	87	5.71	6196	44		TO		-	
epush-6-2	**9.76**	5001	241	571	24197	148		-		-	
epush-6-3	**88**	103k	383		TO			-		-	
epush-10-1	**55.2**	731	268		MC			-		-	
epush-10-2	**392**	64808	845		-			-		-	
epush-10-3		OM			-			-		-	

Table 3: The results on challenging problems modified from conformant problems

clauses of the form one-of$(obj_at(o_i, p_{1,1}), \ldots, obj_at(o_i, p_{n,n}))$, for $i = 1, \ldots, m$. This implies that the number of possible states in the initial belief state is linear in $(n \times n)^m$, i.e., exponential in m—the number of objects—with the base n^2—the number of locations. This explains why all the planners, including CLG and DNF$_{ct}$, scale poorly on m when n is pretty large. In general, however, the scalability of DNF$_{ct}$ on these domains is significantly better than the others thanks to the efficiency of the minimal-DNF representation and the PrAO search algorithm employed by DNF$_{ct}$. On the other hand, DNF$_{ct}$ underperforms CLG on $wumpus$-n. This can be explained as follows. 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.

6.6. Effectiveness of the Pruning

This subsection investigates the effectiveness of the pruning techniques (implemented by the isolation/reconnection propagation procedures) incorporated in PrAO.

Tables 4 and 5 report the detailed results of the performance of DNF$_{ct}$ on a large representative set of problems with/without using the pruning technique. In these tables, columns 2-6 compare the results obtained with the use of the pruning techniques (PrAO left) to those without the use of the pruning (NoPr, right). If the two compared values are equal then the right one is omitted (blank) for simplicity. The question mark represents the unknown value due to time-out or out-of-memory the planner encountered during the search for a solution of the problem. The second column reports, for each problem instance, the search time of DNF$_{ct}$, with the two options, instead of the overall execution time. The reason is that the translation and preprocessing process on each problem instance is independent on the use of the pruning techniques. The last three columns represent the number of reconnected nodes—i.e., the isolated nodes become connected again by the execution of the reconnection propagation procedure—in the solution tree (rgoal), the number of (unexplored) reconnected nodes having chosen to be explored, and the number of nodes ever isolated by the isolation propagation procedure, respectively.

It is easy to see that, for most problems, the application of the pruning results in a significant improvement in the performance of DNF$_{ct}$, e.g., the number of generated and explored nodes (columns 5-6) is smaller and, hence, the execution time (column 2) is shorter. There is a number of large problem instances which DNF$_{ct}$ can solve only with the use of the pruning, e.g., $cball$-3-4, $edisp$-3-5, $epush$-3-5, $epush$-6-3, and $epush$-10-2.

Observe that $ebtcs$-150 is the only problem where the application of the pruning reduces slightly the performance of DNF$_{ct}$. For this problem, the pruning does not help as neither of the two versions of DNF$_{ct}$ (with or without pruning) explored a node among the 11,323 nodes isolated by the pruning and, hence, the two versions of DNF$_{ct}$ generated (resp. explored) the same set of nodes. The time difference (0.06 second), however, is insignificant and one can see that it is also the overhead of the pruning applied in DNF$_{ct}$. This means that the overhead of the pruning is rather small (17.4% of the search time and 13.1% of the overall execution time for the $ebtcs$-150 problem).

The difference in the quality of the solutions (columns 3-4) found by the two versions of DNF$_{ct}$ depends on the domain. However, the difference is insignificant for most problems.

One can see that, for most problems, the isolated nodes form a large portion of the generated nodes (see the last column and the left sub-column of column 5) while the number of reconnected nodes chosen to be explored (column 8) is rather small compared with the number of isolated nodes. This implies that the pruning eliminates a large portion of the search space for most problems. Column 7 indicates that, for most problems, there is a number of reconnected nodes in the solution tree. This confirms the necessity of the reconnection propagation procedure that assures the completeness of the PrAO algorithm.

7. Conclusion

This paper presented the underpinnings of contingent planning along with a standard AND/OR forward search algorithm for contingent planning solutions. The paper extended the minimal-DNF representation of belief states introduced in conformant planning to handle non-deterministic and sensing actions for contingent planning. It then developed the new AND/OR forward search algorithm PrAO, extended from the

Problem	search time		solution size		sol. depth		generated nodes		explored nodes		rgoal	rexp	isolated
	PrAO	NoPr	PrAO	NoPr	PrAO	NoPr	PrAO	NoPr	PrAO	NoPr			
block-7	10.7	11.7	69	74	28	43	42.7k	52.5k	42.1k	52.5k	1	47	704
btcs-150	27.4	27.5	299		150		23.2k		299	448	0	0	22.8k
btnd-150	5.88	5.98	449		152		12.8k	12.8k	595	895	0	0	12k
bts-150	8.56	8.564	299		150		22947		299		0	0	22.5k
ebtcs-150	3.45	3.44	299		150		11.7k		299		0	0	11.3k
ebtnd-150	4.41	4.54	449		152		12.6k	12.7k	597	972	0	0	11.9k
grid-3	0.69	1.05	381	394	67	53	8658	14.4k	5910	14k	1	71	4408
grid-5	11.1	19.9	1337	1327	81	108	142.4k	267.3k	77.6k	266.7k	0	798	88.6k
egrid-3	0.4	0.84	350	392	47	45	5044	11k	3428	10.6k	0	10	2471
egrid-5	8.3	16.3	1469	1195	136	74	112k	245.9k	61k	245.4k	0	580	69.9k
clogistics-5	0.17	0.295	302	312	180	195	4552	6908	1243	6738	6	6	3608
clogistics-7	0.225	0.35	419	233	123	66	6293	8267	1657	8209	8	8	5066
elog.-5	0.21	0.34	301	310	179	193	4552	6908	1243	6738	6	6	3608
elog.-7	0.29	0.41	428	233	136	66	6293	8267	1657	8209	8	8	5061
medpk-99	2.6	26.7	399		102		10.3k	10.4k	399	5347	0	0	9703
medpk-150	8.8	134.3	599		152		22.9k	23.1k	599	11.8k	0	0	22k
unix-4	0.73	0.74	238		179		630	807	294	586	0	0	317
doors-7	0.88	1.45	2193		53		6289	10.7k	4385	10.3k	0	0	2651
doors-9	34.6	69.9	44998	45k	89		131.8k	248.6k	93.6k	242k	0	0	53952
wumpus-5	0.57	0.9	1227	1231	35	41	4106	6431	2930	6213	0	0	1725
wumpus-7	47.6	112.4	30.3k	30.6k	69	74	134.8k	302.3k	96.6k	298.2k	0	0	56365

Table 4: Detailed experimental results for DNF_{ct}: with pruning (PrAO, left) v.s. without pruning (NoPr, right) (columns 2-6). The omitted value (right) is the same as that compared with it (left). *sol. depth*: the depth of the solution; *generated nodes* (*explored nodes*): number of generated (explored) nodes. *rgoal*: number of reconnected nodes in the solution tree; *rexp*: number of reconnected nodes that were explored later; *isol*: number of nodes ever isolated.

Problem	search time		solution size		sol. depth		generated nodes		explored nodes		rgoal	rexp	isolated
	PrAO	NoPr	PrAO	NoPr	PrAO	NoPr	PrAO	NoPr	PrAO	NoPr			
localize-9	0.17	0.36	110		61		372	1037	295	1006	0	1	142
localize-13	1.14	1.93	216		136		877	2066	768	2019	0	0	232
cball-3-3	10	35.8	8030	6324	57	55	25.2k	58.6k	11.6k	58.4k	442	442	18.3k
cball-3-4	340	OM	93.4k	?	73	?	298.4k	?	132.3k	?	5.6k	5.6k	222k
cball-5-2	21	36.7	5165	2999	107	105	18.4k	26.9k	9474	26.7k	417	417	13.1k
cball-8-2	761	1702	27k	15.7k	281	272	123k	265k	61.9k	264k	1.6k	1.6k	87.3k
e1d-3-3	1.07	1.87	3555	3588	88	98	11.8k	14.8k	6023	14.7k	469	472	8940
e1d-5-2	2.51	4.42	2874	2795	128	175	12.1k	20.8k	6064	20.6k	292	292	8846
e1d-5-3	89.6	230	76.1k	73.2k	411	308	314.4k	594.3k	150.9k	593.9k	8.8k	8.8k	238k
e1d-9-2	138	412	31.6k	29.8k	748	682	166.1k	557.2k	81.3k	556.3k	2.2k	2.2k	119k
ecc-40-20	0.55	7.07	466	596	83	190	6746	66k	2255	65.9k	3	3	4815
ecc-75-37	2.43	63.9	903	873	204	196	18k	373k	5979	371k	4	4	12.8k
ecc-119-59	7.96	373	1465	1619	292	310	39.8k	1457k	13k	1449k	5	5	28.5k
edisp.3-3	1.05	2.59	3891	1870	57	57	12.7k	16.4k	6684	16.3k	459	450	9k
edisp.3-5	193	OM	334.5k	?	90	?	1224k	?	590.6k	?	47.5k	49k	940.9k
edisp.5-3	82.3	291	72.4k	14.5k	143	148	300.3k	359.5k	144.4k	359.2k	8.1k	8k	223.4k
edisp.9-2	118	340	22.7k	7029	311	408	121k	260k	59.3k	259k	1.4k	1.4k	85.2k
epush-3-3	0.72	5.99	1249	2605	87	80	6344	54.5k	2830	54.4k	111	172	5143
epush-3-5	120	OM	33.3k	?	149	?	377.4k	?	147.6k	?	3.4k	11.5k	329k
epush-6-2	7	79.1	5030	7644	241	338	25.4k	265.2k	14.4k	264.9k	250	390	15.7K
epush-6-3	354	TO	103k	?	395	?	793.9k	?	413.4k	?	6.3k	13.8k	544k
epush-10-2	319	TO	64.3k	?	845	?	394.2k	?	245.6k	?	2027	3.7k	201k

Table 5: *sol. size* (*sol. depth*): size (depth) of the solution; *generated nodes* (*explored nodes*): number of generated (explored) nodes. *rgoal*: number of reconnected nodes in the solution tree; *rexp*: number of reconnected nodes that were explored later; *isolated*: number of nodes ever isolated.

presented standard algorithm by incorporating the minimal-DNF representation and novel pruning techniques for contingent solutions. While the minimal-DNF-representation provides a compact encoding of belief states and fast state computation, PrAO's goal is to minimize the number of nodes generated and explored during the search. Both the minimal-DNF-representation and the pruning allow for the development of a complete contingent planner. The correctness of most theoretical results, except several trivial ones, has been provided in the paper.

The presented techniques have been implemented in a new contingent planner, called DNF$_{ct}$, which was experimentally evaluated 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.

For a better understanding of the effectiveness of the pruning technique incorporated in PrAO, the work furthered the empirical study by comparing the performance of DNF$_{ct}$ with a variant of it where the pruning techniques were not applied. The experimental results showed that the pruning eliminates a large portion of the search space and improves the performance significantly on most problems. There are also a number of large problem instances which only the DNF$_{ct}$ with the use of the pruning techniques can solve.

Although DNF$_{ct}$ exhibits great performance and scalability, several open questions still remain. For example, there are a few problems on which disjunctive representation appears to be unsuitable (e.g., $wumpus$, $dispose$,...) or the heuristic function employed in DNF$_{ct}$ does not work well (e.g., $block$, the variants of $logistics$, and $grid$). As such, for problems rich in disjunctive information, another representation might be needed (e.g., conjunctive normal form). Furthermore, the study of more sophisticated effective heuristic schemes would enhance the performance of DNF$_{ct}$ or any contingent planners—that employ the PrAO search with an efficient representation method—greatly such that they would be able to solve a wide range of real-life applications.

References

[1] Alexandre Albore, Héctor Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1623–1628, 2009.

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

[3] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Mbp: a model based planner. In *Proc. IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information, Seattle, August*, 2001.

[4] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. pages 52–61. AAAI Press, 2000.

[5] Ronen Brafman and Jörg Hoffmann. Conformant planning via heuristic forward search: A new approach. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, Whistler, Canada.

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

[7] D. Bryce, S. Kambhampati, and D. Smith. Planning Graph Heuristics for Belief Space Search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[8] Eric A. Hansen and Shlomo Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

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

[10] Jörg Hoffmann and Ronen I. Brafman. Contingent planning via heuristic forward search with implicit belief states. 2005.

[11] Jörg Hoffmann and Ronen I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7):507–541, 2006.

[12] Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302, 2001.

[13] H. Palacios and H. Geffner. From Conformant into Classical Planning: Efficient Translations that may be Complete Too. In *Proceedings of the 17th International Conference on Planning and Scheduling*, pages 264–271, 2007.

[14] H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

[15] M. Peot and D.E. Smith. Conditional nonlinear planning. In *Proc. of 1st International Conference on AIPS*, pages 189–197, 1992.

[16] L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.

[17] J. Rintanen. Complexity of planning with partial observability. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 2004.

[18] S.T. To, E. Pontelli, and T.C. Son. A Conformant Planner with Explicit Disjunctive Representation of Belief States. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, pages 305–312. AAAI, 2009.

[19] S.T. To, T.C. Son, and E. Pontelli. A New Approach to Conformant Planning using CNF. In *Proceedings of the 20th International Conference on Planning and Scheduling (ICAPS)*, pages 169–176, 2010.

[20] S.T. To, T.C. Son, and E. Pontelli. On the Use of Prime Implicates in Conformant Planning. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[21] S.T. To, T.C. Son, and E. Pontelli. Contingent Planning as AND/OR Forward Search with Disjunctive Representation. In *Proceedings of the 21th International Conference on Planning and Scheduling (ICAPS-2011)*, pages 258–265, 2011.

[22] D. Warren. Generating conditional plans and programs. In *Proceedings of Summer Conference on AI and Simulation of Behavior (AISB)*, 1976.