Conformant Planning for Domains with Constraints — A New Approach

Tran Cao Son and Phan Huy Tu
Computer Science Department
New Mexico State University
Las Cruces, NM 88003, USA
{tson,tphan}@cs.nmsu.edu

Michael Gelfond and A. Ricardo Morales
Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
{mgelfond,ricardo}@cs.ttu.edu

Abstract

The paper presents a pair of new conformant planners, CPA^{pc} and CPA^{ph}, based on recent developments in theory of action and change. As an input the planners take a domain description $\mathcal{D}$ in action language $\mathcal{AL}$ which allows state constraints (non-stratified axioms), together with a set of CNF formulae describing the initial state, and a set of literals representing the goal. We propose two approximations of the transition diagram T defined by $\mathcal{D}$. Both approximations are deterministic transition functions and can be computed efficiently. Moreover they are sound (and sometimes complete) with respect to T. In its search for a plan, an approximation based planner analyses paths of an approximation instead of that of T. CPA^{pc} and CPA^{ph} are forward, best first search planners based on this idea. We compare them with two state-of-the-art conformant planners, KACMBP and Conformant-FF (CFF), over benchmarks in the literature, and over two new domains. One has large number of state constraints and another has a high degree of incompleteness. Our planners perform reasonably well in benchmark domains and outperform KACMBP and CFF in the first domain while still working well with the second one. Our experimental result shows that having an integral part of a conformant planner to deal with state constraints *directly* can significantly improve its performance, extending a similar claim for classical planners in (Thiebaux, Hoffmann, & Nebel 2003).

Keywords: Planning, Knowledge Representation, Reasoning about action and change

Introduction and Motivation

In recent years, several conformant planners have been developed for solving planning problems in the presence of incomplete information about the initial state. These planners can be divided into two groups. In the first group, the planning problem is translated into an equivalent problem in a more general setting which can be solved by off-the-shelf software systems. Belonging to this group are the SAT-based planner $\mathcal{C}$-PLAN (Castellini, Giunchiglia, & Tacchella 2003), QBF planner (Rintanen 2000), the model checking planner CMBP (Cimatti & Roveri 2000) (or its newer version KACMBP (Cimatti, Roveri, & Bertoli 2004)), and answer set programming based planners (Eiter *et al.* 2003; Son, Tu, & Baral 2004). In the second group, the focus has been on developing efficient search strategies, good heuristics, or new search algorithms (Brafman & Hoffmann 2004; Bryce & Kambhampati 2004; Bonet & Geffner 2001; Kurien, Nayak, & Smith 2002; Petrick & Bacchus 2002; Smith & Weld 1998). While the majority of planners in the first group allow state constraints[1] to be part of the planning problem specification, other planners, including the most recent additions to the set of conformant planners (Brafman & Hoffmann 2004; Bryce & Kambhampati 2004), do not.

Most of the planners in the second group deal with constraints by compiling them away, i.e., planning problems with constraints are compiled into planning problems without constraints. This practice has several disadvantages (Thiebaux, Hoffmann, & Nebel 2003): (i) the problem representation can become unnatural and unreadable due to the extra actions and fluents; (ii) allowing state constraints significantly increases the expressive power of the representation language. The authors in (Thiebaux, Hoffmann, & Nebel 2003) also suggested a compilation schema that compiles away state constraints and produces a new problem whose size is linear to the size of the original domain. Unfortunately, this may not be enough. The following simple example illustrates this point.

Example 1 (Dominos Domain) Suppose that we have n dominos standing on a line in such a way that if one domino falls then the domino on its right also falls. There is also a ball hanging close to the leftmost domino. Touching the ball will cause the leftmost domino to fall. Initially, the ball stays still and whether or not the dominos are standing is unknown. The goal is to have the rightmost domino to fall. Obviously, swinging the ball is the only plan to achieve this goal, no matter how big n is.

The problem can be easily expressed by a theory with a set of objects $1, \ldots, n$ denoting the dominos from left to right and a single action $swing$ that causes $down_1$ (the leftmost domino falls) to be true, and $n-1$ state constraints $down_i \Rightarrow down_{i+1}$ representing the fact that $down_{i+1}$ is true if $down_i$ is true. The goal is to have $down_n$ become true.

According to the compilation suggested in (Thiebaux, Hoffmann, & Nebel 2003), for each axiom $down_i \Rightarrow$

[1] We use the term state constraints (or constraints) to refer to PDDL axioms or static causal laws in action languages.

$down_{i+1}$, we introduce a new action e_i whose effect is $down_{i+1}$ and whose precondition is $down_i$. Clearly, under this compilation, the plan to achieve the goal is the sequence of actions $[swing, e_1, \ldots, e_{n-1}]$.

The main problem with this compilation is that the plan length increases with the number of objects. Even when it is only linear to the size of the original problem, it proves to be challenging for planners following this approach. We tested this simple problem with some of the state-of-the-art planners that do not support state constraints. Most returned no solution after 30 minutes when n>500. □

In this paper we present a pair of new conformant planners based on recent developments in theory of action and change. As an input such planners take a domain description $\mathcal{D}$ in action language $\mathcal{AL}$ which allows state constraints, together with a set of CNF formulae describing the initial situation, and a set of literals representing the goal. In our implementation, the latter two are translated into a set of initial partial states and a goal partial state correspondingly. The planners' search spaces are defined by two transition diagrams whose nodes are sets of partial states, called approximations of the transition diagram T of $\mathcal{D}$. These proposed approximations have two important properties: (i) they are deterministic and their transition functions can be computed efficiently, (ii) conformant plans can be found by analyzing paths of the corresponding approximation. Futhermore, if the set of initial partial states is singleton then this reduces the complexity of the conformant planning problem to **NP**-complete, comparing to Σ_P^2-complete (Turner 2002). Although approximation based planners are in general incomplete, our planners are powerful enough to solve all literature benchmarks used in our experiment. Given that the underlying heuristic functions do sometime stumble in certain domains, we view the theoretical incompleteness as a reasonable price for gaining efficiency.

To summarize, the main contributions of the paper are:

- two sound approximations for reasoning about actions and their effects in the presence of axioms and incomplete information about the initial situation. To the best of our knowledge, such approximations have been developed only for theories with incomplete information about the initial state and sensing actions that do not contain axioms (Son & Baral 2001). In a recent paper, an approximation has been developed but only for very limited class of theories (Gelfond & Morales 2004).
- two conformant planners whose performance is comparable with state-of-the-art conformant planners in several domains. The key component of the planner is the module for computing the approximations.
- two new domains for testing conformant planners; the *domino domain* (Exp. 1) is rich with constraints and the *cleaner domain* (later) has a high degree of incompleteness in the initial state; these domains seem to be difficult for current state-of-the-art conformant planners.

Background

We begin with a short review of the basic definitions of the language $\mathcal{AL}$ from (Baral & Gelfond 2000) and a fixpoint characterization for domains with state constraints. The alphabet of a domain consists of a set of action names **A** and a set of fluent names **F**. A (fluent) literal is either a fluent $f \in \mathbf{F}$ or its negation $\neg f$. A *domain description* (or a domain) $\mathcal{D}$ is a set of laws of the following forms:

$$a \textbf{ causes } l \textbf{ if } \psi \quad (1)$$

$$l \textbf{ if } \psi \quad (2)$$

$$\textbf{impossible } a \textbf{ if } \psi \quad (3)$$

where $a \in \mathbf{A}$ is an action, l is a fluent literal, and ψ is a set of literals. (1) is called a *dynamic causal law*, describing the explicit effect of action a. It says that if a is performed in a state wherein ψ holds then l will hold in every successor state. (2), called a *state constraint*, says that in any state in which ψ holds, then so does l. (3) is called an impossibility law which states that a cannot be executed when ψ holds. Given a domain $\mathcal{D}$, $\mathcal{D}^d$ (resp. $\mathcal{D}^s$) denotes the set of dynamic causal laws (resp. state constraints in $\mathcal{D}$). Observe that there is no syntactical restriction on state constraints.

Example 2 The domino domain in Example 1, denoted by $\mathcal{D}_1$, can be represented by a domain with the dynamic causal law $swing$ **causes** $down_1$ and the set of constraints $\{down_{i+1}$ **if** $down_i \mid 1 \leq i \leq n-1\}$. □

We now introduce notations that will be used throughout the paper. For a fluent f, $\neg\neg f = f$. For a set of literals σ, $\neg\sigma = \{\neg l \mid l \in \sigma\}$. A literal l (resp. set of literals γ) holds in a set of literals σ if $l \in \sigma$ (resp. $\gamma \subseteq \sigma$); l (resp. γ) possibly holds in σ if $\neg l \notin \sigma$ (resp. $\neg\gamma \cap \sigma = \emptyset$). A set of literals σ satisfies a constraint (2) if it holds that $\psi \subseteq \sigma$ implies $l \in \sigma$. σ is *closed* under $\mathcal{D}^s$ if it satisfies every $r \in \mathcal{D}^s$. By $Cl_{\mathcal{D}}(\sigma)$, we denote the smallest set of literals that contains σ and is closed under $\mathcal{D}^s$. An *interpretation* I of **F** is a set of literals such that for every $f \in \mathbf{F}$, $\{f, \neg f\} \cap I \neq \emptyset$ and $\{f, \neg f\} \setminus I \neq \emptyset$. A *state* of $\mathcal{D}$ is an interpretation of **F** closed under $\mathcal{D}^s$. An action a is *executable* in s if there exists no dynamic causal law (3) such that ψ holds in s.

Given a domain $\mathcal{D}$, for an action a and a state s such that a is executable in s, let $e(a, s) = \{l \mid a$ **causes** l **if** $\psi \in \mathcal{D}^d, \psi \subseteq s\}$. In essence, $e(a, s)$ denotes the direct effects of a. We define the set of possible successor states after executing a in s, denoted by $Res^c_{\mathcal{D}}(a, s)$, as follows.

Definition 1 ((McCain & Turner 95)) *Let $\mathcal{D}$ be a domain description and a be an action that is executable in a state s. A state s' is called a* possible successor *of s after the execution of a if $s' = Cl_{\mathcal{D}}(e(a, s) \cup (s \cap s'))$.*

The set of possible successors of s after the execution of a is denoted by $Res^c_{\mathcal{D}}(a, s)$.

As an example, the state $s^0 = \{\neg down_i \mid 1 \leq i \leq n\}$ represents the fact that all dominos are standing. Furthermore, $e(swing, s^0) = \{down_1\}$ and $Res^c_{\mathcal{D}_1}(swing, s^0) = \{\{down_i \mid 1 \leq i \leq n\}\}$.

We say that a domain is *inconsistent* if $Res^c_{\mathcal{D}}(a, s) = \emptyset$ for some action a and state s such that a is executable in s. In the rest of the paper, we are only interested in consistent domains. Intuitively, Definition 1 says that if the agent is currently in state s then after executing a it will reach one of

the states in $Res^c_{\mathcal{D}}(a,s)$. In the presence of incomplete information, the agent does not always know what exact state it is currently in. It might need to consider a set $\mathcal{W}$ of possible states rather than a single one. In this case, the set of possible successor states, $Res^c_{\mathcal{D}}(a,\mathcal{W})$, is defined as

- $Res^c_{\mathcal{D}}(a,\mathcal{W}) = \emptyset$ if $Res^c_{\mathcal{D}}(a,s) = \emptyset$ for some $s \in \mathcal{W}$;
- $Res^c_{\mathcal{D}}(a,\mathcal{W}) = \bigcup_{s\in\mathcal{W}} Res^c_{\mathcal{D}}(a,s)$, otherwise.

The set of states reached after executing an action sequence $\alpha = \langle a_1;\ldots;a_n\rangle$ from a set of states $\mathcal{W}$ is defined next.

$$Res^c_{\mathcal{D}}(\alpha,\mathcal{W}) = \begin{cases} \mathcal{W} \text{ if } n = 0 \\ Res^c_{\mathcal{D}}(a_2;\ldots;a_n, Res^c_{\mathcal{D}}(a_1,\mathcal{W})) \text{ if } n \geq 1 \end{cases}$$

We say that a literal l holds after the execution of α in $\mathcal{W}$, denoted by $(\mathcal{D},\mathcal{W}) \models l$ **after** α, if $Res^c_{\mathcal{D}}(\alpha,\mathcal{W}) \neq \emptyset$ and $l \in s$ for every state $s \in Res^c_{\mathcal{D}}(\alpha,\mathcal{W})$.

By a *partial state* of $\mathcal{D}$ we mean a consistent collection of fluent literals closed under $\mathcal{D}^s$. Partial states are denoted by (possibly indexed) Greek letters. A state s containing a partial state δ is called a *completion* of δ. By $ext(\delta)$ we denote the set of all completions of δ. For a set of partial states Δ, let $ext(\Delta) = \cup_{\delta\in\Delta} ext(\delta)$.

Let $\mathcal{D}$ be a domain description, Δ^0 be a set of partial states, and δ^f be a partial state. We say that a sequence of actions α is a *conformant plan* achieving δ^f from Δ^0 if $(\mathcal{D}, ext(\Delta^0)) \models l$ **after** α for every $l \in \delta^f$. We define:

Definition 2 Given a domain description $\mathcal{D}$, a set of partial states Δ^0, a parial state δ^f. The *conformant planning problem* is the problem of computing a tractable conformant plan achieving the goal δ^f from Δ^0.

The *1-conformant planning problem* is the conformant planning problem restricted to the case where $|\Delta^0| = 1$.

Abusing the notation, we often refer to $\langle \mathcal{D}, \Delta^0, \delta^f\rangle$ as a planning problem whenever there is no confusion possible.

Two Approximations of $Res^c_{\mathcal{D}}$

Given a conformant planning problem, most of search-based planners look for solutions by exploring the belief state[2] space whose size is double exponential in the number of fluents. Adding to this, determining what is true/false after one action is executed in the presence of incomplete information is a co-**NP** complete problem even when state constraints are not present (Baral, Kreinovich, & Trejo 2000). As such, we begin our quest for building a conformant planner by looking for ways to reduce the complexity of the task. We achieve this goal by developing two sound (but incomplete) approximations of the function $Res^c_{\mathcal{D}}$ which we denote by $Res^{ph}_{\mathcal{D}}$ and $Res^{pc}_{\mathcal{D}}$ (ph and pc stand for "possibly holds" and "possibly changes", respectively). We sometime write $Res^a_{\mathcal{D}}$ whenever we would like to refer to either $Res^{ph}_{\mathcal{D}}$ and/or $Res^{pc}_{\mathcal{D}}$. Each approximation is a function that maps pairs of partial states and actions into partial states. Both are deterministic and can be computed efficiently. Moreover, both reduce the size of the state space to single exponential to the number of fluents.

Given an action a and a partial state δ, we will now define $Res^a_{\mathcal{D}}(a,\delta)$, an approximation of what will hold after the execution of a in δ. Before presenting the formulae defining $Res^a_{\mathcal{D}}$, let us observe that each possible successor state s' in Definition 1 can be divided into three parts: (i) $e(a,s)$ contains the direct effects of a; (ii) $s \cap s'$ contains what remains unchanged (because of the inertial law); and (iii) the set of the indirect effects of a. Any formulation of the $Res^a_{\mathcal{D}}$ should account for these three components.

To specify the direct effects of an action we take the view of a skeptical reasoner. Given a partial state δ and an action a, we define $e(a,\delta) = \{l \mid a$ **causes** l **if** $\psi \in \mathcal{D}^d$, and $\psi \subseteq \delta\}$ and $mc(a,\delta) = \{l \mid a$ **causes** l **if** $\psi \in \mathcal{D}^d$, and $\neg\psi \cap \delta = \emptyset\}$. Intuitively, $e(a,\delta)$ (resp. $mc(a,\delta)$) consists of the direct effects (reps. possible direct effects) of a when it is executed in a state in which δ holds. As an example, for the action $swing$ in $\mathcal{D}_1$, we have that $e(swing,\delta) = mc(swing,\delta) = \{down_1\}$ for every δ.

The main difficulty in characterizing $Res^a_{\mathcal{D}}$, however, lies in specifying the second component, i.e., what remains unchanged by the inertial law. Different ways of defining this set lead to different approximations of $Res^c_{\mathcal{D}}$. We next introduce two possibilities.

Approximation Based on What Possibly Holds

We will now define $Res^{ph}_{\mathcal{D}}$ which approximates $Res^c_{\mathcal{D}}$ based on what possibly holds. Here, we view the inertial part as a set of literals whose negations cannot possibly hold. Given an action a and a partial state δ such that a is executable in δ, a literal l possibly holds in a successor state, say δ', if one of the following happens.

- a might cause l to hold, i.e., $l \in mc(a,\delta)$;
- $\neg l$ does not hold in δ and there exists no dynamic causal law (1) s.t. ψ holds in δ, i.e., $\neg l \notin (\delta \cup e(a,\delta))$;
- there exists a constraint (2) s.t. ψ possibly holds in δ'.

$Res^{ph}_{\mathcal{D}}(a,\delta)$ conservatively defines the second component as the set of all literals whose negations cannot possibly hold. Formally, let $ph(a,\delta) = Cl_{\mathcal{D}}(mc(a,\delta) \cup \{l \mid \neg l \notin (\delta \cup e(a,\delta))\})$, we define

$$Res^{ph}_{\mathcal{D}}(a,\delta) = Cl_{\mathcal{D}}(e(a,\delta) \cup \{l \mid l \notin \neg ph(a,\delta)\})$$

if $Cl_{\mathcal{D}}(e(a,\delta) \cup \{l \mid l \notin \neg ph(a,\delta)\})$ is consistent; otherwise, $Res^{ph}_{\mathcal{D}}(a,\delta) = \bot$. We will discuss some properties of $Res^{ph}_{\mathcal{D}}$ after the definition of the second approximation.

Approximation Based on What Possibly Changes

While $Res^{ph}_{\mathcal{D}}$ approximates the inertial part by looking at what might hold in the successor partial state, $Res^{pc}_{\mathcal{D}}$ looks at what might change. It resembles $Res^c_{\mathcal{D}}$ by assuming that the result is known, say δ'. That is, assume that $Res^{pc}_{\mathcal{D}}(a,\delta) = \delta'$. We say that a literal l is *possibly changed* after the execution of a in δ if it does not belong to δ but possibly holds in δ'. We denote the set of possibly changed literals by $pc(a,\delta,\delta')$. Observe that a literal l possibly changes its value if $l \notin \delta$ and

- a might directly cause l, i.e., $l \in mc(a,\delta)$; or
- there exists a constraint (2) s.t. ψ possibly holds in δ' and contains at least one literal that possibly changes.

This leads us to define $pc(a,\delta,\delta') = \cup_{i=0}^{\infty} pc^i(a,\delta,\delta')$ where

$pc^0(a,\delta,\delta') = mc(a,\delta) \setminus \delta$,

$pc^{i+1}(a,\delta,\delta') = (pc^i(a,\delta,\delta') \cup \Omega_i) \setminus \delta$ for $i \geq 0$ with

[2] A belief state is a set of states.

Ω_i=$\{l \mid l$ **if** $\psi \in \mathcal{D}^s, \neg\psi \cap \delta'$=$\emptyset$, and $\psi \cap pc^i(a,\delta,\delta')$≠$\emptyset\}$. The definition of $Res^{pc}_{\mathcal{D}}$ rests on the following observations: (i) δ' must contain $Cl_{\mathcal{D}}(e(a,\delta))$; (ii) $Cl_{\mathcal{D}}(\delta' \cup (\delta \setminus \neg pc(a,\delta,\delta')))$ holds in every succesor state resulting from executing a in a state satisfying δ; and (iii) for the sequence of partial states $\delta_1, \delta_2, \ldots$ where δ_1=$Cl_{\mathcal{D}}(e(a,\delta))$, and δ_{i+1}=$Cl_{\mathcal{D}}(\delta_i \cup (\delta \setminus \neg pc(a,\delta,\delta_i)))$ for i≥1, it holds that (a) $\delta_i \subseteq \delta_{i+1}$; and (b) δ_i holds in every successor state resulting from executing a in a state satisfying δ; and (c) this sequence converges to a partial state δ^*. We therefore define $Res^{pc}(a,\delta)$ as

$$Res^{pc}_{\mathcal{D}}(a,\delta) = \delta^*$$

if δ^* is consistent; and $Res^{pc}_{\mathcal{D}}(a,\delta) = \bot$ otherwise. We will now discuss some properties of the approximations.

Properties of the Approximations

Notice that in the definitions of $Res^a_{\mathcal{D}}$, we take into account the three components mentioned earlier: (i) the direct effect of a: $e(a,\delta)$; (ii) the inertial part: $\{l \mid l \notin \neg ph(a,\delta)\}$ or $\delta \setminus \neg pc(a,\delta,\delta')$ (iii) the indirect effect of a: those generated by the operator $Cl_{\mathcal{D}}$. It is easy to see that $Res^a_{\mathcal{D}}(a,\delta)$ is uniquely defined, i.e., the functions Res^a are deterministic.

The definitions of $Res^c_{\mathcal{D}}(a,\mathcal{W})$ and $\models$ are extended to define $Res^a_{\mathcal{D}}(a,\Delta)$ and $\models^a$ in a straightforward way. We omit them here to save space.

Example 3 Consider the following domain description $\mathcal{D}_2$ defined over the set of fluents $\{f,g,h,k,p,q\}$.

a **causes** f	a **causes** g **if** k	g **if** f,h
g **if** $f,\neg h$	k **if** f	p **if** g,q

Suppose that we perform a in $\Delta = \{\delta\}$, where $\delta = \{\neg f, \neg g, \neg p, \neg q\}$. Intuitively, we would expect that afterward f should be true (as a direct effect of a); k and g should be true (because of "k **if** f", and "g **if** f,h" and "g **if** $f,\neg h$" respectively); and p,q should be false (because of inertia).

We have that $e(a,\delta) = \{f\}$ and $mc(a,\delta) = \{f,g\}$.
$X = \delta \cup e(a,\delta) = \{\neg f, \neg g, \neg p, \neg q, f\}$
$Y = \{l \mid \neg l \notin X\} = \{\neg g, \neg p, \neg q, h, \neg h, k, \neg k\}$
$Z = mc(a,\delta) \cup Y = \{f, g, \neg g, \neg p, \neg q, h, \neg h, k, \neg k\}$
$ph(a,\delta)$=$Cl_{\mathcal{D}_2}(Z)$=$\{f, g, \neg g, \neg p, \neg q, h, \neg h, k, \neg k\}$
Hence, $Res^{ph}_{\mathcal{D}_2}(a,\delta) = \{f, \neg p, \neg q, k\}$.
For Res^{pc}, $\delta_1 = Cl_{\mathcal{D}_2}(e(a,\delta)) = \{f,k\}$ and
$pc^0(a,\delta,\delta_1) = \{f,g\}$, $\Omega_0 = \{k,g,p\}$
$pc^i(a,\delta,\delta_1) = \{f,g,k,p\}$, $\Omega_i = \{k,g,p\}$ for $i \geq 1$.
This leads to $\delta_2 = \{f,k,\neg q\}$. Repeating this computation with δ_2, we get δ_3=δ_2. So, $Res^{pc}_{\mathcal{D}_2}(a,\delta)$=$\{f,k,\neg q\}$.

As a result, we have $Res^{pc}(a,\Delta) = \{\{f,k,\neg q\}\}$ and $Res^{ph}_{\mathcal{D}_2}(a,\Delta) = \{\{f,\neg p,\neg q,k\}\}$.

Observe that both $Res^{ph}_{\mathcal{D}_2}(a,\Delta)$ and $Res^{pc}_{\mathcal{D}_2}(a,\Delta)$ draw $f,k,\neg q$ as expected; furthermore, the former can draw $\neg p$, whereas the latter cannot. But none of them could draw g. Nevertheless, if Δ was $\{\delta \cup \{h\}, \delta \cup \{\neg h\}\}$, then both could draw g. Some other entailments w.r.t. the approximation semantics are: $(\mathcal{D}_2,\Delta) \models^a f$ **after** a; $(\mathcal{D}_2,\Delta) \not\models^a g$ **after** a; $(\mathcal{D}_2,\Delta) \not\models^a \neg g$ **after** a. □

We now show some properties of the approximations. The next theorem shows that $\models^a$ is sound w.r.t. $\models$.

Theorem 1 (Soundness w.r.t. $\models$) *Let $\mathcal{D}$ be a consistent domain description, α be a sequence of actions, Δ be a set of partial states, and l be a literal. Then,*

$(\mathcal{D},\Delta) \models^a l$ **after** α implies that $(\mathcal{D}, ext(\Delta)) \models l$ **after** α.

Example 3 shows that the approximations are not complete w.r.t. $Res^c_{\mathcal{D}}$. Apart from this, $Res^{pc}(a,\delta) \subseteq Res^{ph}(a,\delta)$. Our conjecture is that this relationship holds. However, we have not yet found a formal proof for this property. We also show that

Theorem 2 *Given a domain description $\mathcal{D}$, for any partial state δ and action a, $Res^a_{\mathcal{D}}(a,\delta)$ can be computed in polynomial time in the size of the domain.*

This allows us to prove the following result.

Theorem 3 *The 1-conformant planning problem w.r.t. the approximation semantics is* **NP***-complete.*

We have, among other things, extended both approximations to consider concurrent actions and non-deterministic actions. We also identified a large class of domains in which $\models^a$ is equivalent to $\models$. These result will be presented in the complete version of this paper. In the next section, we describe our initial experiments with these approximations in the development of conformant planners.

Conformant Planning using Approximations

We first implemented an answer set programming based conformant planner using Res^{ph}, which we will refer to as CPASP. In this sense, CPASP is similar to DLV^k (Eiter *et al.* 2003), CMBP (Cimatti & Roveri 2000), $\mathcal{C}$-PLAN (Castellini, Giunchiglia, & Tacchella 2003), QBF (Rintanen 2000). CPASP allows parallel actions. We tested CPASP against DLV^k, CMBP, and $\mathcal{C}$-PLAN because these planners do allow state constraints and are similar in spirit of CPASP. Our results show that CPASP is competitive with these planners in most of the domains from the literature. Due to space limit, we omit here the detailed encoding of CPASP and the experimental result. The main weakness of CPASP is that it does not solve problems with disjunctive information about the initial state. In other words, CPASP can handle only 1-conformant planning problems (i.e., for $|\Delta^0| = 1$).

To overcome this weakness, we implemented a pair of planners, called CPA^{pc} and CPA^{ph}, based on $Res^{pc}_{\mathcal{D}}$ and $Res^{ph}_{\mathcal{D}}$ respectively, in C++. By convention, we will write CPA whenever we want to refer to both planners or one of them when the distinction between them is not important. CPA employs the best first search strategy with repeated state avoidance and the number of fulfilled subgoals as the heuristic function. The main module of CPA is for computing the $Res^a_{\mathcal{D}}$ function.

CPA accepts problems encoded using the rules of the forms (1)-(3). The initial state is specified by statements of the form **initially** ϕ (where ϕ is in the CNF form). As mentioned ealier, in our implementation, these CNFs will be translated into a set of initial partial states, e.g., the set $\{$**initially** $f \vee g$, **initially** $p \vee q\}$ results in $\Delta^0 = \{Cl_{\mathcal{D}}(\{f,p\}), Cl_{\mathcal{D}}(\{f,q\}), Cl_{\mathcal{D}}(\{g,p\}), Cl_{\mathcal{D}}(\{g,q\})\}$. The goal is encoded using statements of the form **goal** ϕ where ϕ is a set of literals.

We compare CPA with three planners CFF, KACMBP, and POND. These planners were selected for the following reasons. First, CFF and KACMBP are — to the best of our knowledge — the current fastest conformant planners in most of the benchmark domains in the literature. Second, CFF is superior to other state-of-the-art conformant planners like GPT (Bonet & Geffner 2001), MBP (Cimatti & Roveri 2000) (see (Brafman & Hoffmann 2004)). Third, KACMBP, a heuristic guided, symbolic model checking based conformant planner supporting state constraints, is known to outperform DLV^k and $\mathcal{C}$-PLAN in many domains in the literature (see (Cimatti, Roveri, & Bertoli 2004)); Finally, POND is a new addition to the set of conformant planners that implemented several interesting heuristics.

We prepare two test-suites. The first one consists of typical conformant planning domains including the Bomb-in-the-toilet (Bomb), the Ring, and Logistics domains. The first domain was chosen because both CFF and KACMBP work well with it (e.g. CFF can scale up the Bomb domain with multiple toilets to 100 packages and 100 toilets within a minute). The latter two were chosen because the experiments in (Brafman & Hoffmann 2004) showed that CFF is good at the Logistics problem but not the Ring, while KACMBP is good at Ring but has poor performance with Logistics.

In the Bomb domain, we experimented with $10, 20, 50, 100$ packages and $t = 1, 5, 10$ toilets.

The Logistics domain used in our tests is described in (Brafman & Hoffmann 2004) and distributed together with the CFF distribution. We did experiments with 5 problems, corresponding to $l = 2, 3, 4$ and $c = p = 2, 3$, where l, c, and p are the numbers of locations per city, cities, and packages respectively (only Logistics(4,2,2) is not available).

In the Ring domain, one can move in a cyclic fashion (either forward or backward) around a n-room building to lock windows. Each room has a window and the window can be locked only if it is closed. The uncertainty is that the initial state of windows is unknown. The goal is to have all windows locked. A possible conformant plan is to perform a sequence of actions *forward*, *close*, *lock* repeatedly. In this domain, we tested with n =2,5,10, and 20.

Three domains in the first test suite, however, do not contain many state constraints. Most of the constraints in our encodings are aimed at expressing multivalued variables as boolean fluents[3]. To see how good these planners are in dealing with domains rich in constraints, in the second test suite, we include the Domino domain. We tested the domain with eight problems corresponding to $n = 10, 50, 200, 100, 500, 1000, 2000, 5000$, where n is the number of dominos. Note that we encoded these problems for POND and CFF following the compilation procedure in (Thiebaux, Hoffmann, & Nebel 2003).

The second domain included in the second test suite is the Cleaner domain. It is a modified version of the Ring domain. The difference is that the robot is moving in a linear fashion rather than in a cyclic fashion and instead of locking the window, the robot has to clean p objects in each room. Initially, the robot is in the first room and does not know whether or not objects are cleaned. The goal is to have all objects cleaned. While the Domino domain exposes a richness in constraints, the Cleaner domain provides a high degree of uncertainty in the initial state. We tested the domain with 6 problems corresponding to $n = 2, 5$ and $p = 10, 50, 100$.

All experiments were made on a 2.4 GHz CPU, 768MB RAM machine, running Slackware 10.0 operating system. Time limit is set to half an hour. The testing results[4] for two test suites are shown in Tables 1–2. Times are shown in seconds; 'TO", "AB", and "NA" indicate that time limit is exceeded, that the planner stopped abnormally, and that the domain is not applicable, respectively.

Domains	KACMBP	CFF	POND	CPA^{pc}	CPA^{ph}
Bomb(10,1)	19 / 0.01	19 / 0.05	19/2.61	19 / 0.01	19 / 0.02
Bomb(20,1)	39 / 0.05	39 / 0.17	/AB	39 / 0.06	39 / 0.13
Bomb(50,1)	99 / 0.51	99 / 5.33	/AB	99 / 0.87	99 / 1.83
Bomb(100,1)	199 / 3.89	199 / 121.80	/AB	199 / 7.63	199 / 14.8
Bomb(10,5)	15 / 0.09	15 / 0.07	/AB	15 / 0.03	15 / 0.07
Bomb(20,5)	35 / 0.30	35 / 0.16	/AB	35 / 0.17	35 / 0.37
Bomb(50,5)	95 / 1.66	95 / 4.70	/AB	95 / 2.74	95 / 4.82
Bomb(100,5)	195 / 6.92	195 / 113.95	/AB	195 / 24.17	195 / 36.90
Bomb(10,10)	10 / 0.30	10 / 0.05	/AB	10 / 0.05	10 / 0.13
Bomb(20,10)	30 / 0.97	30 / 0.13	/AB	30 / 0.35	30 / 0.81
Bomb(50,10)	90 / 5.39	90 / 4.04	/AB	90 / 5.34	90 / 9.470
Bomb(100,10)	190 / 35.83	190 / 102.56	/AB	190 / 43.79	190 / 65.43
Ring(2)	5 / 0.00	7 / 0.06	5/0.16	5 / 0.00	5 / 0.00
Ring(5)	14 / 0.00	45 / 63.67	15/39.37	15 / 0.15	15 / 0.21
Ring(10)	29 / 0.02	/ TO	/TO	30 / 3.25	30 / 5.03
Ring(15)	44 / 0.04	/ TO	/TO	45 / 21.40	45 / 34.94
Ring(20)	59 / 0.15	/ TO	/TO	60 / 84.02	60 / 141.33
Ring(25)	74 / 0.32	/ TO	/TO	75 / 244.02	75 / 420.20
Logistic(2,2,2)	14 / 0.19	16 / 0.03	/NA	12 / 0.75	12 / 1.16
Logistic(2,3,3)	34 / 355.96	24 / 0.06	/NA	106 / 120.86	106 / 181.95
Logistic(3,2,2)	17 / 2.10	20 / 0.06	/NA	96 / 48.15	96 / 74.16
Logistic(3,3,3)	40 / 29.80	34 / 0.12	/NA	/ TO	/ TO
Logistic(4,3,3)	/ TO	37 / 0.14	/NA	/ TO	/ TO

Table 1: Conformant Planning Benchmarks

As can be seen in Table 1, in the Bomb domain, KACMBP is the best in general. CPA is competitive with CFF in most of problems and outperforms CFF in some others. For example, CPA^{pc} took 7.63 seconds to solve BMTC(100,1), while CFF took 121.80 seconds. However, CFF seems to have no problem when the number of toilets increases, while there is a significant increase in the amount of time for KACMBP. The change in the amount of time for CPA is more reasonable than that for KACMBP. For example, with a fixed number of packages 100, when the number of toilets increase from 5 to 10, the amount of solving time for CFF even decreases, while that for KACMBP about 5 times increases and CPA's is just doubled. POND can solve Bomb(10,1) only.

The Ring domain is really problematic for CFF. As explained in (Brafman & Hoffmann 2004), it is because of the lack of informativity of the heuristic function in the presence of non-unary effect conditions and the problem with checking repeated states. CFF can solve only the first two problems within the time limit. Again, KACMBP is the best. CPA is much better than CFF and POND but is not competitive with KACMBP.

[3]Unlike POND, CPA, and CFF, an advantage of KACMBP is that it allows multivalued fluents. Thus, it seems to perform well with numeric domains like Ring, Cube, and Square, etc. (see (Cimatti, Roveri, & Bertoli 2004) for the performance of KACMBP over these domains)

[4]In each cell, the first number is the length of the solution. The second number is the time taken by the planner to find the solution.

In the Logistic domain, both KACMBP and CPA had difficulty in finding plans. Although KACMBP is better than CPA, its performance is far from that of CFF which solved each problem in less than one second. We believe that the poor performance of CPA lies in the not-so-good heuristic function (which is reflected in the plan's length).

Table 2 shows the testing results for the second test-suite. As expected, using the mentioned compilation procedure, CFF has poor performance in the Domino domain. Within the time limit, KACMBP can only solve problems with $n < 500$ but CPA has no problem with $n = 5000$.

Domains	KACMBP	CFF	POND	CPA^{pc}	CPA^{ph}
Domino(10)	23 / 0.01	10 / 0.05	10/1.72	1 / 0.00	1 / 0.00
Domino(50)	163 / 0.27	50 / 4.44	/TO	1 / 0.00	1 / 0.01
Domino(100)	376 / 2.56	/ AB	/TO	1 / 0.02	1 / 0.03
Domino(200)	852 / 29.10	/ AB	/TO	1 / 0.06	1 / 0.09
Domino(500)	/ TO	/ AB	/TO	1 / 0.35	1 / 0.59
Domino(1000)	/ TO	/ AB	/TO	1 / 1.38	1 / 2.48
Domino(2000)	/ TO	/ AB	/TO	1 / 5.38	1 / 10.74
Domino(5000)	/ TO	/ AB	/TO	1 / 36.48	1 / 62.68
Cleaner(2,10)	21 / 0.08	21 / 0.07	/AB	21 / 0.03	21 / 0.06
Cleaner(2,20)	41 / 0.62	41 / 0.15	/AB	41 / 0.13	41 / 0.38
Cleaner(2,50)	101 / 13.55	101 / 0.80	/AB	101 / 1.71	101 / 5.39
Cleaner(2,100)	201 / 185.39	201 / 5.72	/AB	201 / 13.97	201 / 44.09
Cleaner(5,10)	56 / 0.10	54 / 0.24	/AB	54 / 0.41	54 / 0.86
Cleaner(5,20)	106 / 7.82	104 / 0.85	/AB	104 / 2.32	104 / 5.52
Cleaner(5,50)	256 / 227.82	254 / 14.36	/AB	254 / 30.83	254 / 85.20
Cleaner(5,100)	/ TO	/ AB	/AB	504 / 239.08	504 / 688.39

Table 2: Domains with Constraints and High Degree of Incompleteness

CPA has a relatively good performance in the Cleaner domain. It can solve $Cleaner(5, 100)$ within the time limit. The returned plan has 504 actions. CFF is very good at this domain and outperforms CPA and KACMBP (POND cannot solve any problems of these). Unfortunately, it cannot solve the last problem since the maximum length of a plan is exceeded. CPA outperforms KACMBP in most problems in this domain.

As stated, our planner is sound but not complete, i.e., theoretically speaking, CPA cannot solve some planning problems, even when the initial state is complete. To make sure our approach can cover a broader spectrum of practical planning problems, we also tested CPA with classical planning problems. The first domain considered is the Block World with five problems described in (Eiter *et al.* 2003). We then tested with problems in the Rovers domain[5]. Five problems, different from each other in the numbers of way points, rovers, cameras, rock and soil samples, and objectives, were experimented with. It turns out that CPA can solve all those problems but did not perform well in the Blocks World domain. We suspect that our heuristic is not good enough to guide the planner in this domain.

Discussion and Conclusion

We have presented a pair of new conformant planners, CPA^{pc} and CPA^{ph}, which deal directly with state constraints. Their performance is comparable with state-of-the-art conformant planners over typical benchmark domains as well as over newly invented domains. Due to the simple heuristic used in the implementation of CPA, we believe that the good performance of CPA lies in the use of the approximations. The development of CPA demonstrates that research in reasoning about action and change can positively impact the development of practical planners. Although CPA yields good performance, there are a number of issues that need to be investigated. On the implementation side, we would like to improve CPA's performance by testing it with different heuristics used in other planners. We would also like to find plans with parallel actions and/or minimal plans. Theoretically, we would like to investigate the relationship between Res^{ph} and Res^{pc} and to find a better approximation that allows for limited reasoning by cases for use with CPA. We also would like to strengthen characteristics of domains in which the approximations are complete[6].

Acknowledgement: The first two authors were partially supported by NSF grants EIA-0220590 and HRD-0420407.

[5] http://planning.cis.strath.ac.uk/competition/

[6] It is worth noting that the current characteristics seem to cover most problems found in the literature. This result will be available in the complete version of this paper.

References

Baral, C.; and Gelfond, M. 2000. Reasoning agents in dynamic domains. In Minker, J,. ed., LBAI, 257–279.

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

Bonet, B., and Geffner, H. 2001. GPT: a tool for planning with uncertainty and partial information. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information, 82–87.

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. ICAPS-04, 355–364.

Bryce, D., and Kambhampati, S. 2004. Heuristic Guidance Measures for Conformant Planning. ICAPS-04, 365–375.

Castellini, C.; Giunchiglia, E.; and Tacchella, A. 2003. SAT-based Planning in Complex Domains: Concurrency, Constraints and Nondeterminism. *AIJ* 147:85–117.

Cimatti, A., and Roveri, M. 2000. Conformant Planning via Symbolic Model Checking. *JAIR* 13:305–338.

Cimatti, A. et al. 2004. Conformant Planning via Symbolic Model Checking and Heuristic Search. *AIJ* 159:127–206.

Eiter, T. et al. 2003. A Logic Programming Approach to Knowledge State Planning, II: The DLV^{K} System. *AIJ* 144, 157–211.

Gelfond, M., and Morales, R. 2004. Encoding conformant planning in a-prolog. In *Proceedings of DRT'04*, LNCS.

Kurien, J.; Nayak, P. P.; and Smith, D. E. 2002. Fragment-based conformant planning. In *AIPS*, 153–162.

McCain, N., and Turner, H. 95. A causal theory of ramifications and qualifications. IJCAI, 1978–1984.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. AIPS-02, 212–222.

Rintanen, J. 2000. Constructing conditional plans by a theorem prover. *JAIR* 10:323–352.

Smith, D. & Weld, D. 1998. Conformant Graphplan. *AAAI*.

Son, T., and Baral, C. 2001. Formalizing sensing actions - a transition function based approach. *AIJ* 125(1-2):19–91.

Son, T. et al. 2004. Planning with Sensing Actions and Incomplete Information using Logic Programming. LPNMR, 261–274.

Thiebaux, S.; Hoffmann, J.; and Nebel, B. 2003. In Defense of PDDL Axioms. IJCAI'03.

Turner, H. 2002. Polynomial-length planning spans the polynomial hierarchy. JELIA'02, 111–124.