On Building a Competitive Comformant Planner

Tran Cao Son

Department of Computer Science
New Mexico State University
tson@cs.nmsu.edu

1 Invited Abstract

In this talk, I will detail the development of CpA(H), a competitive conformant planner, that won the Conformant Planning Category in the International Planning Competition 2006. Lessons learned and the influence of logic programming in our development of the planner will also be discussed.

Conformant Planning with Dis unctive Initial States: Design and Development of an Ef cient Planner

D-V. Tran, H-K. Nguyen, E. Pontelli, T.C. Son

Department of Computer Science
New Mexico State University
vtran | knguyen | epontell | tson@cs.nmsu.edu

Abstract. The paper illustrates the design and development of a competitive conformant planner. The planner builds on the theoretical foundations of approximation-based planning to enable a compact representation of the possible states. The novelty of the proposed approach is the realization that the description of the (incomplete) initial state is often based on *constraints* (e.g., expressed through PDDL's `or` and `oneof` clauses); the paper illustrates how such constraints can be reasoned upon to reduce the size of the search space. The reasoning process is implemented in the form of transformations of the problem specification within the planner. This, along with approximations and the use of combined heuristics, leads to enhanced efficiency and scalability, outperforming state-of-the-art conformant planners on several benchmark suites.

1 Introduction and Motivation

Conformant planning is the problem of finding a sequence of actions that achieves the goal from every possible initial state of the world [14]. One of the main difficulties encountered in the process of determining a conformant plan is the high degree of uncertainty, due to the potentially large number of possible initial states of the problems.

The *Planning Domain Definition Language (PDDL)* introduces two constructs to express incomplete knowledge about the initial state of the world: *mutual-exclusion* statements (expressed using `oneof`-clauses) and *disjunctive* statements (expressed using `or`-clauses). Frequently, `oneof`-clauses are used to specify the possible initial states and `or`-clauses are used to eliminate infeasible states. Because of this, the number of possible initial states depends mainly on the number and the size of the `oneof`-clauses and these are often exponential in the number of constants present in the problem instances. For example, three out of six domains in the 2006 planning competition have this property (Table 1).

Instance	Cons/States	Instance	Cons/States
comm-15	$35/2^{16}$	coins-20	$17/9 \times 8^6$
comm-20	$85/2^{21}$	coins-25	$39/10^{20}$
comm-25	1402^{26}	coins-30	$45/10^{25}$
sortnet-10	$11/2^{11}$	sortnet-15	$2^{16}/16$

Table 1. Number of Constants/Possible Initial States

Effective methodologies and data structures are required to deal with the large number of possible initial states. Some conformant planners, such as POND [6] and KACMBP [7], employ a BDD representation of belief states, while others, such as CFF [4], adopt a CNF representation. These types of encodings avoid dealing directly with the exponential number of states, but they require extra work in determining the truth value of certain fluents after the execution of a sequence of actions in the initial belief state. For instance, CFF needs to make a call to a SAT-solver with the initial state and the sequence of actions; other planners need to recompute the BDD representation, which could also be an expensive operation. Observe that the problem of determining the truth value of a proposition after the execution of a single action in a belief state is co-NP complete [1].

An alternative approach to deal with the large number of possible initial states is used by the planners `cf2cs(ff)` and CPA [12, 17], and further investigated in their successors `t0` and CPA+ [13, 16]. This approach relies on an *approximation semantics* in reasoning with incomplete information [15]. The planners `cf2cs(ff)` and `t0` reduce the number of possible initial states to one by introducing additional propositions, transforming the original problem to a classical planning problem, and using FF, a classical planner [8], to find solutions. On the other hand, CPA and CPA+ reduce this number by dividing them into groups and using the intersection of each group as its representative during planning.

CPA+ and `t0` implement the idea of approximations differently. While CPA+ could be seen as a standard heuristic search forward planner, `t0` follows a translational approach. The performance of CPA+ depends on its heuristic function and its ability to approximate the initial belief state to a manageable set of partial states. On the other hand, the performance of `t0` largely depends on the performance of FF. `t0` was the winner of the 2006 planning competition.

In this paper, we describe the design and implementation of a competitive conformant planner. The proposed planner[1] expands the idea of approximation-based conformant planning, introducing novel techniques to significantly enhance efficiency and scalability. We explore the problem of engineering an approximation-based planner that can avail of modern data structures and heuristic functions. A cornerstone of our approach is viewing the description of the initial state not just as a passive characterization of a collection of states, but as a collection of constraints; by reasoning on such constraints, we discover ways to transforms the problem specification, enabling drastic reductions in the size of the search space. The resulting planner outperforms the state-of-the-art in conformant planning on large pool of benchmarks, including the problems from the latest planning competition.

2 Problem Representation

Following the notation in [12], we describe a *problem specification* as a tuple $P = \langle F, O, I, G \rangle$, where F is a set of propositions, O is a set of actions, I and O describe the initial state of the world and the goal respectively. A *literal* is either a proposition $p \in F$ or its negation $\neg p$. $\bar{\ell}$ denotes the complement of a literal ℓ and is defined by

[1] Named CPA(H) to recognize its roots in the CPA+ system.

$\bar{\ell} = \neg\ell$ where $\neg\neg p{=}p$ for $p{\in}F$. For a set of literals L, $\overline{L} = \{\bar{\ell} \mid \ell \in L\}$. A conjunction of literals is often represented by a set.

A set of literals X is *consistent* if there exists no $p \in F$ such that $\{p, \neg p\} \subseteq X$. A *state* s is a consistent and *complete* set of literals, i.e., s is consistent, and for each $p \in F$, either $p \in s$ or $\neg p \in s$. A *belief state* is a set of states. A *partial state* is a consistent set of literals. A *cs-state* is a set of partial states. A set of literals X satisfies a literal ℓ (resp. a set of literals Y) iff $\ell \in X$ (resp. $Y \subseteq X$).

Each action a in O is associated with a precondition ϕ (denoted by $pre(a)$) and a set of conditional effects of the form $\psi \rightarrow \ell$ (also denoted by $a : \psi \rightarrow \ell$), where ϕ and ψ are sets of literals and ℓ is a literal.

The initial state of the world is described by $I = I^d \cup I^o \cup I^r$ where I^d is a set of literals, I^o is a set of `oneof`-clauses of the form $\texttt{oneof}(\phi_1, \ldots, \phi_n)$ and I^r is a set of `or`-clauses of the form $\texttt{or}(\phi_1, \ldots, \phi_n)$, where each ϕ_i is a set of literals. A `oneof`-clause indicates that the ϕ_i's are mutually exclusive, while an `or`-clause is a disjunctive normal form (DNF) representation of a formula. A set of literals X satisfies $\texttt{oneof}(\phi_1, \ldots, \phi_n)$ if there exists some $1{\leq}i{\leq}n$ s.t. $\phi_i \subseteq X$ and for every $j \neq i$, $1{\leq}j{\leq}n$, $\overline{\phi_j} \cap X{\neq}\emptyset$. X satisfies $\texttt{or}(\phi_1, \ldots, \phi_n)$ if there exists some $1{\leq}i{\leq}n$ s.t. $\phi_i \subseteq X$. $ext(I)$ denotes the set of all states satisfying I^d, every `oneof`-clause in I^o, and every `or`-clause in I^r. E.g., if $F{=}\{g, f\}$ and $I{=}\{g, \texttt{oneof}(f, \neg f)\}$ then $ext(I) = \{\{g, f\}, \{g, \neg f\}\}$.

G can contain literals or `or`-clauses. Given a `oneof`-clause or an `or`-clause o, we write $L \in o$ to denote that L is an element of o and $lit(o) = \bigcup_{L \in o}(L \cup \bar{L})$.

3 A Competitive Conformant Planner: Design

The planner, called CPA(H), is composed of two modules. The first module (*Preprocessor*) is a static analyzer that performs a number of transformations of the problem specification. Along with a grounder (which also applies standard simplifications, such as forward reachability), the preprocessor applies some novel transformations (`oneof`-clause combination and goal splitting) aimed at drastically reducing the size of the search space. The second module (*Planning engine*) is a heuristic search engine implementing forward planning.

3.1 Design of the Planning Engine

Theoretical Foundations Given a state s and an action a, a is executable in s if $pre(a) \subseteq s$. The set of effects of a in s, denoted by $e_a(s)$, is defined by: $e_a(s) = \{l \mid \psi \rightarrow l \text{ is an effect of } a, \psi \subseteq s\}$. The execution of a in a state s results in a successor state $succ(a, s)$ which is defined by: $succ(a, s) = s \cup e_a(s) \setminus \overline{e_a(s)}$ if a is executable in s; and $succ(a, s) = failed$, otherwise.

$succ$ is extended to define $succ^*$, which computes the result of the execution of an action in a belief state, as follows.

$$succ^*(a, S) = \begin{cases} \{succ(a, s) \mid s \in S\} \\ \quad \text{if } a \text{ is executable in every } s \in S \\ failed \quad \text{otherwise} \end{cases} \tag{1}$$

Finally, we can define the function $\widehat{succ}$ to compute the final belief state resulting from the execution of a plan:

- $\widehat{succ}([a_1, \ldots, a_n], S) = S$ if $n = 0$, and
- $\widehat{succ}([a_1, \ldots, a_n], S) = succ^*(a_n, \widehat{succ}([a_1, \ldots, a_{n-1}], S))$ if $n > 0$.

Several heuristic search-based conformant planners (e.g. CFF, POND), employ $\widehat{succ}$ in plan computation, using $S_0 = ext(I)$ as the initial belief state. An action sequence α is a *solution* of P iff $\widehat{succ}(\alpha, S_0) \neq failed$ and G is satisfied in every state belonging to $\widehat{succ}(\alpha, S_0)$.

The notion of approximation used in CPA+, `cf2cs(ff)`, and `t0` has been originally proposed in [15]. The original intuition behind approximation is to approximate sets of possible states by a single partial state thus, reducing the complexity of reasoning w.r.t. using all possible states. It is characterized by a function ($succ_A$) that maps an action and a partial state to a partial state. The *possible effects* of a in a partial state δ are given by

$$pc_a(\delta) = \{l \mid \psi \rightarrow l \text{ is an effect of } a, \overline{\psi} \cap \delta = \emptyset\}. \quad (2)$$

The successor partial state from the execution of a in δ is defined by $succ_A(a, \delta) = (\delta \cup e_a(\delta)) \setminus \overline{pc_a(\delta)}$ if a is executable in δ; and $succ_A(a, \delta) = failed$, otherwise.

Similarly to $succ^*$ and $\widehat{succ}$, $succ_A$ can be extended to define $succ_A^*$ (mapping cs-states to cs-states) and $\widehat{succ_A}$ for computing the result of the execution of an action sequence starting from a cs-state. The notion of a solution of P w.r.t. the approximation is extended accordingly. Our planner uses $\widehat{succ_A^*}$ in its search for plans.

Observe that, in general, reasoning using approximations is incomplete. Completeness can be gained by identifying appropriate partitions $\{\Delta_1, \ldots, \Delta_k\}$ of $ext(I)$ such that $\bigcup_{i=1}^{k} \Delta_i = ext(I)$, $\Delta_i \cap \Delta_j = \emptyset$ for each $i \neq j$, and we have that for each formula φ and sequence of actions α, $\widehat{succ_A}(\alpha, \{\delta_1, \ldots, \delta_k\})$ entails φ iff $\widehat{succ}(\alpha, ext(I))$ entails φ, where δ_i is the intersection of the states in Δ_i. Research has been conducted to provide sufficient syntactical conditions to identify valid partitions based on the identification of fluents that should be explicitly distinguished in different partitions (see, e.g., [16]).

euristic Search The $succ_A^*$ function is used in the context of a planning algorithm which implements forward planning using a traditional best-first heuristic search.

The proposed planner enables the user to choose among different heuristics; as discussed in the experimental section, we observed that sustained better performance can be achieved by using combinations of heuristics, improving the ability of discriminating between states in the priority queue (as employed in other systems as well [6]). The basic heuristics employed are:

- *The cardinality heuristic:* we prefer belief states that have a smaller cardinality. In other words, $h_{card}(\Sigma) = |\Sigma|$ where Σ is a belief state. Note that we use this heuristic in a forward fashion, and this is different from its use in [2, 5]. The intuition behind this is that planning with complete information is "easier" than planning with incomplete information and a lower cardinality implies a lower degree of uncertainty.

- *The relaxed graphplan heuristic:* for a belief state Σ, we define $h_{rgp}(\Sigma) = \sum_{\delta \in \Sigma} d(\delta)$, where $d(\delta)$ is the well-known sum heuristic value given that the initial state is $\delta \cup \{\neg p \mid p \in F,\ p \notin \delta,\ \neg p \notin \delta\}$ [11].
- *The number of satisfied subgoals:* denoted by $h_{gc}(\Sigma)$.

We investigate the following combination of these heuristics: $h_{css}(\Sigma) = (h_{card}(\Sigma), h_{gc}(\Sigma), h_{rgp}(\Sigma))$; heuristic measures are compared according to their lexicographic order.

3.2 Design of the Preprocessor

Standard Transformations Key to our analysis is the notion of *dependence* between actions and propositions similar to the notion of dependence between actions and literals explored in [16]. We denote with P a planning problem.

De nition 1. *An action a* depends *on a literal ℓ if*

1. *$\ell \in pre(a)$, or*
2. *there exists an effect $a : \phi \rightarrow h$ in P and $\ell \in \phi$, or*
3. *there exits an action b that depends on ℓ and a depends on some of the effects of b, i.e., b depends on ℓ and there exists $b : \phi \rightarrow h$ such that a depends on h.*

By $preact(\ell)$ we denote the set of actions depending on ℓ.

Intuitively, the fact that a depends on ℓ indicates that the truth value of ℓ could influence the result of the execution of a. For a set of literals L, $preact(L) = \bigcup_{\ell \in L} preact(\ell)$.

De nition 2. *Two literals ℓ and ℓ' are* distinguishable *if $\ell \neq \ell'$ and there is no action that depends on both ℓ and ℓ', i.e., $preact(\ell) \cap preact(\ell') = \emptyset$.*

Obviously, the distinguishable relation is symmetric and irreflexive. Two set of literals L_1 and L_2 are distinguishable if $preact(L_1) \cap preact(L_2) = \emptyset$.

The dependence between a literal and an action, often used in *reachability analysis*, is defined next.

De nition 3. *A literal ℓ* depends *on an action a if* (1) *$a : \psi \rightarrow \ell$ is in P; or* (2) *there exists an action b such that $b : \psi \rightarrow \ell$ is in P and there exists some ℓ' in ψ or in $pre(b)$ s.t. ℓ' depends on a. We denote with $deps(a)$ the set of literals that depend on a.*

Intuitively, ℓ depends on a implies that ℓ may be achieved by executing a. $postact(\ell)$ denotes the set of actions which ℓ depends on, i.e., $postact(\ell)$=$\{a \mid \ell \in deps(a)\}$.

De nition 4. *Two literals ℓ and ℓ' are* independent *if $\ell \neq \ell'$ and there exists no action that both ℓ and ℓ' depend on, i.e., $postact(\ell) \cap postact(\ell') = \emptyset$.*

The preprocessor starts its operations with a number of basic normalization steps, aimed at reducing the number of propositions and the number of actions present in the problem specification. In particular, it implements a traditional *forward reachability* simplification (to detect propositions whose value cannot be changed and actions that cannot be triggered w.r.t. the initial state) and a symmetrical *goal relevance* (removing actions that cannot contribute to the goal).

Combination of `oneof` clauses The idea of this technique is based on the *non*-interaction between actions and propositions in different sub-problems of a conformant planning problem. We illustrate this idea in the next example.

Example 1. Let $P = \langle \{f, g, h, p, i, j\}, O, I, G\rangle$ where $I = \{\texttt{oneof}(f, g), \texttt{oneof}(h, p), \neg i, \neg j\}$, $G = i \wedge j$, and $O = \{\, a : f \rightarrow i \; c : h \rightarrow j \; b : g \rightarrow i \; d : p \rightarrow j \,\}$. It is easy to see that the sequence $\alpha = [a, b, c, d]$ is a solution of P. Furthermore, the search should start from the initial belief state consisting of the four states:

$$\{f, \neg g, h, \neg p, \neg i, \neg j\} \; \{\neg f, g, h, \neg p, \neg i, \neg j\}$$
$$\{f, \neg g, \neg h, p, \neg i, \neg j\} \; \{\neg f, g, \neg h, p, \neg i, \neg j\}$$

Let P' be the problem obtained from P by replacing I with I', where $I' = \{\texttt{oneof}(f \wedge h, g \wedge p), \neg i, \neg j\}$.

We can see that α is also a solution of P'. Furthermore, each solution of P' is also a solution of P. This transformation in interesting since the initial belief state now consists only of two states: $\{f, \neg g, h, \neg p, \neg i, \neg j\}$ and $\{\neg f, g, \neg h, p, \neg i, \neg j\}$. I.e., the number of states in the initial belief state that a conformant planner has to consider in P' is 2, while it is 4 in P. This transformation is possible because the set of actions that are "activated" by f and g is disjoint from the set of actions that are "activated" by h and p, i.e., $preact(\{f, g\}) \cap preact(\{h, p\}) = \emptyset$.

The above example shows that different `oneof`-clause can be combined into a single `oneof`-clause, which effectively reduces the size of the initial state that a planner needs to consider in its search for a solution. Theoretically, if the size of the two `oneof`-clauses in consideration is m and n, then it is possible to achieve a reduction in the number of possible partial states from $m \times n$ to $\max(m, n)$. Since in many problems the size of the `oneof`-clauses increases with the number of objects, being able to combine the `oneof`-clauses could provide a significant advantage for the planner.

Definition 5. *Let $P = \langle F, O, I, G\rangle$ be a planning problem. Two* `oneof`*-clauses o_1 and o_2 are* combinable *if (*i*) $lit(o_1) \cap lit(o_2) = \emptyset$; and (*ii*) $lit(o_1)$ is distinguishable from $lit(o_2)$. where $lit(o)$ denote the union of the set of literals occurring in o and its complements*

For example, the `oneof`-clauses in Ex. 1 are combinable. Let $o_1 = \texttt{oneof}(L_1, \ldots, L_n)ando_2 = \texttt{oneof}(S_1, \ldots, S_m)$. Assume that $n \geq m$. A *combination* of o_1 and o_2, denoted by $o_1 \oplus o_2$ (or $o_2 \oplus o_1$) is the clause

$$\texttt{oneof}(L_1 \wedge S_1, \ldots, L_m \wedge S_m, L_{m+1} \wedge S_1, \ldots, L_n \wedge S_1)$$

Intuitively, a combination of o_1 and o_2 is a `oneof`-clause whose elements are pairs obtained by composing one element of o_1 with exactly one element of o_2.

Proposition 1. *Let $P = \langle F, O, I, G\rangle$ be a planning problem, where G is a conjunction of literals and o_1 and o_2 are two combinable* `oneof`*-clauses in P. Let $P' = \langle F, O, I', G\rangle$, where I' is obtained from I by replacing o_1 and o_2 by $o_1 \oplus o_2$. Every solution of P' is also a solution of P and vice versa.*

Observe that the above proposition may not hold if P contains disjunctive goals, as shown next.

Example 2. Let $P = \langle \{q, g, h, p, i, j\}, O, I, G\rangle$ where

$$I = \{\texttt{oneof}(h, g), \texttt{oneof}(p, q), \neg i, \neg j\} \text{ and } G = \texttt{or}(i, j)$$

and O consists of $a : p, \neg q \rightarrow i$, $c : p, q \rightarrow i$, $b : g, \neg h \rightarrow j$, and $d : g, \neg h \rightarrow j$.

It is easy to check that $\texttt{oneof}(h, g)$ and $\texttt{oneof}(q, p)$ are combinable. Let P' be the problem obtained from P by replacing I with $I' = \{\texttt{oneof}(g \wedge q, h \wedge p), \neg i, \neg j\}$. Then, $[a, b]$ is a solution of P' but not a solution of P.

The *combinable* notion can be generalized as follows.

De nition 6. *A set of* oneof*-clauses* $\{o_1, \ldots, o_k\}$ *is combinable if* o_i *and* o_j *are combinable for each* $1 \leq i \neq j \leq k$.

Let $\oplus(o_1, \ldots, o_k)$ be the shorthand for $(((o_1 \oplus o_2) \oplus \ldots) \oplus o_k)$. Proposition 1 can be generalized as follows.

Proposition 2. *Let* $P = \langle F, O, I, G\rangle$ *be a planning problem, where* G *is a conjunction of literals. Let* $\{o_1, \ldots, o_k\}$ *be a combinable set of* oneof*-clauses in* P. *Let* $P' = \langle F, O, I', G\rangle$, *where* I' *is obtained from* I *by replacing* $\{o_1, \ldots, o_k\}$ *with* $\oplus(o_1, \ldots, o_k)$. *We have that each solution of* P' *is a solution of* P *and vice versa.*

We implemented a greedy algorithm, whose running time is polynomial in the size of P, for detecting sets of combinable oneof-clauses and replacing them with their corresponding combination. This is possible since testing if ℓ and ℓ' are distinguishable can be done in polynomial time in the size of P, and the number of pairs that need this test is quadratic in the number of propositions.

Goal Splitting Reducing the size of the initial state only helps the planner to start the search. It does not necessarily imply that the planner can find a solution. In this section, we present another technique, called *goal-splitting*, which can be used in conjunction with the combination of oneof to deal with large planning problems. This technique can be seen as a variation of the goal ordering technique in [9] and it relies on the notion of dependence proposed in Def. 4. The key idea is that if a problem P contains a subgoal whose truth value cannot be negated by the actions used to reach the other goals, then the problem can be decomposed into smaller problems with different goals, whose solutions can be combined to create a solution of the original problem. This is illustrated in the following example.

Example 3. Consider the problem P of Example 1. It is easy to see that the two goals i and j are independent and P can be decomposed into two sub-problems $P_1 = \langle F, O_1, I, i\rangle$ and $P_2 = \langle F, O_2, I_2, j\rangle$ where $O_1 = \{a : f \rightarrow i,\ \ b : g \rightarrow i\}$ and $O_2 = \{c : h \rightarrow j,\ \ d : p \rightarrow j\}$ with the following property: if α is a solution of P_1 and β is a solution of P_2 where $I_2 = \widehat{succ_A}(\alpha, I_1)$, then $\alpha; \beta$ is a solution of P.[2]

Let us start with a definition capturing the condition that allows the splitting of goals.

[2] $\alpha; \beta$ denotes the concatenation of two sequences of actions.

De nition 7. *Let $P = \langle F, O, I, G\rangle$ be a planning problem and let $\ell \in G$. We say that ℓ is G-separable if, for each $\ell' \in G \setminus \{\ell\}$ we have that $\bar{\ell}$ and ℓ' are independent.*

Proposition 3. *Let $P = \langle F, O, I, G\rangle$ be a planning problem and let ℓ be G-separable. Let $P_\ell = \langle F, postact(\ell), I, \ell\rangle$ and α be a solution of P_ℓ. Let $P_{G\setminus\{\ell\}} = \langle F, postact(G \setminus \{\ell\}), I', G \setminus \{\ell\}\rangle$, where $I' = \widehat{succ_A}(\alpha, I)$, and β be a solution of $P_{G\setminus\{\ell\}}$. Then, $\alpha;\beta$ is a solution of P.*

The proof is trivial, since $postact(G \setminus \{\ell\})$ does not contain any action that can make $\bar{\ell}$ true.

On the other hand, it is easy to see that not every plan of P can be split into two parts α and β such that α is a solution of P_ℓ and β is a solution of $P_{G\setminus\{\ell\}}$. We can prove, however, that for each plan γ of P, there is a plan α for P_ℓ and a plan β for $P_{G\setminus\{\ell\}}$ such that γ is a permutation of $\alpha;\beta$. This provides a weak form of completeness.

We note that the splitting proposed in Prop. 3 can be improved by also splitting the propositions and initial states into different theories. We have implemented a generalized version of Prop. 3 to split a problem into a sequence of problems. This implementation runs in polynomial time in the size of P.

4 Implementation Considerations

The preprocessor has been implemented as a Prolog program. The program maps the input PDDL theory to a collection of Prolog clauses. This mapping nicely avoids the need of explicitly grounding the problem specification a priori. The transformations are implemented as fixpoint computations on the Prolog clauses representing the problem specification.

The planning engine has been implemented as a C++ program, running on a Linux (Athlon 64, 4Ghz), gcc 4.2.1 version, with STL library. A partial state is implemented as a set (a basic data structure in STL) of literals.

The engine implements a best first search over the search space of cs-states. Each cs-state is a data structure consisting of a set of partial states, a plan to reach that cs-state, and the heuristic values: h_{card}, h_{gc}, and h_{rpg}. A modified version of the algorithm presented in [10] is implemented to compute h_{rpg}.

$succ^*_A$ is used to compute the next cs-state. A hash table (resp. priority queue) is used to store the visited (resp. unvisited) cs-states. A special module is developed to compute the initial cs-state, which consists of the set of initial partial states. Each initial partial state δ satisfies the following conditions: a) $\{p, \neg p\} \cap \delta \neq \emptyset$ for each proposition p appears in I; b) $I^d \subseteq \delta$; c) for each $\texttt{oneof}(\phi_1, \ldots, \phi_n) \in I^o$, there exists an i such that $\phi_i \subseteq \delta$ and for all $j \neq i$, $\overline{\phi_j} \cap \delta \neq \emptyset$; d) for each $\texttt{or}(\phi_1, \ldots, \phi_n) \in I^r$, there exists an i such that $\phi_i \subseteq \delta$; e) δ is consistent. Choosing to implement the initial cs-state as a set (of the set of initial partial states) makes the computation of the successor cs-state (the result of $succ^*_A$) easier. The main disadvantage of this choice is that the size of the initial cs-state can be exponential in the size of the number of object constants in the problem. This is the reason why reducing the size of the initial cs-state is critical to our planner.

5 Experimental Evaluation

The experimental evaluation has been performed using several benchmark suites i.e., problems from the IPC-5 (or I5) and the IPC-6 (I6) planning competitions, challenging (C) problems proposed in [13], and several other domains from previous planning competitions. The benchmark suite for each domain is listed in Table 5. Due to lack of space, we omit the detailed description of the actual benchmarks, that have been drawn from the existing literature. We also report only a subset of the complete experimental results due to limited space (full results will be made available through a linked technical report). Time is in seconds, TO denotes time-out (30 min), AB denotes out of memory, and BM denotes benchmark suite.

Table 2 summarizes some results aimed at evaluating the impact of the transformations; the three columns indicate execution times and the length of the first plan found; we can observe that the improvement in performance is often significant; it occasionally comes at the price of a longer plan.

Problem	NoTransf.	`oneof`	goal-splitting
coins-05	0.02/13	0.024/19	0.03/14
coins-10	1.33/35	0.66/129	1.52/43
coins-15	/AB	13.54/391	/AB
coins-20	/AB	33.39/621	/AB
comm-05	0.90/48	0.31/60	0.20/35
comm-10	61.14/87	3.98/190	22.40/65
comm-15	/AB	15.82/327	/AB
uts-05	34.43/115	34.43/115	1.06/43
uts-10	36.14/130	36.14/130	21.21/87
uts-20	53.88/286	53.88/286	35.91/138
uts-30	152.07/177	152.07/177	18.06/74
dispose-4-2	1.30/72	0.395/59	0.78/76
dispose-4-3	43.00/93	0.36/78	19.14/111
dispose-8-2	412.70/272	10./234	368.03/284
dispose-8-3	/AB	487.28/1187	/AB
push-4-2	1.11/58	1.00/133	1.84/96
push-4-3	34.66/265	2.04/251	46.68/141
push-8-2	282.55/444	128.21/979	/AB
push-8-3	/AB	454.80/1811	/AB

Table 2. Impact of `oneof`-combination and goal-splitting

Table 3 reports the execution times of the planner using different heuristics. The column t_S indicates the time for preprocessing. Although the results are mixed for small instances, $h_{css(\Sigma)}$ comes ahead for large instances.

Table 4 compares the execution times and plan lengths for the proposed planner and other three state-of-the-art systems for conformant planning (`t0`, CFF, and POND, all run with default parameters according to their documentation). Table 5 reports the number of instances each planner can solve.

6 Discussion

We now discuss the question of whether the proposed techniques can be applied to other planning systems.

Observe that a combination of several `oneof`-clauses is a `oneof`-clause, whose elements are conjunctions of literals, which can be represented by a set of `oneof`-clauses

Problem	t_S	CPA(H) h_{card}	CPA(H) h_{gc}	CPA(H) h_{rgp}	CPA(H) $h_{css(\Sigma)}$
bwl-02	0.13	0.262/26	/TO	0.064/33	0.217/41
bwl-03	0.18	7.668/198	/TO	2.219/145	73.188/312
coins-15	0.49	7.449/423	15.874/551	0.387/191	5.920/329
coins-20	0.66	25.51/756	24.413/722	0.902/195	20.239/481
comm-15	3.64	0.496/96	0.148/95	0.094/95	0.124/95
comm-20	141	2.739/240	0.993/239	0.994/239	0.976/239
comm-25	1081	18.74/389	3.355/389	3.674/389	3.249/389
sortnet-05	0.11	0.054/13	16.35/13	0.036/12	0.023/12
sortnet-10	0.24	12.537/39	/TO	3.270/39	4.205/39
sortnet-15	0.52	/TO	/TO	/TO	313.044/65
uts-10	0.93	21.21/87	/TO	17.567/89	9.602/80
uts-20	0.89	35.91/138	/TO	12.596/150	36.314 /125
uts-30	0.86	18.06/74	/TO	346.467/103	31.609/94
d-4-3	.61	. 3.94/288	4.83/369	0.95/314	3.80/288
d-8-1	47.3	1.95/143	124.71/1229	24.12/725	1.86/137
d-8-2/C	53.6	386.68/1328	600.77/2298	135.8/1494	391.44/1328
d-10-1	285	7.65/213	/AB	148.7/1489	7.69/213
push-4-3	.65	40.45/1176	340.23/959	1.0/225	288.21/847
push-8-1/C	52.4	3.824/184	/AB	27.25/468	3.97/184
push-10/C	304	23.04/414	/AB	/AB	23.48/414
1-d-4-3	.69	24.8/64	/AB	81.57/108	25.12/64
1-d-8-1	47.9	9.91/340	/AB	/AB	9.22/340
1-d-10-1	288	36/568	/AB	/AB	33.81/568
lng-8-1-1	50.4	2.45/94	284.73/5547	/AB	1.78/94
lng-8-2-1	57.8	0.97/94	31.79/563	68.45/287	0.97/94
lng-8-1-2	59.5	73.14/125	/AB	/AB	72.65/125

Table 3. Comparison between heuristics

and a set of disjunctions eliminating the unwanted combinations. We tested the effectiveness of the `oneof`-simplification on POND and CFF. Table 6 shows the results of our experiment with the planner POND in the `comm` and `coins` domains where the `oneof`-combination is applicable. As we can see, the performance of POND improves in these problems, and the improvement is more significant when the size of the problem is large. This technique helps POND to scale up but its impact is not as great as in CPA(H): POND can solve more problems in the `comm` domain. The problems `comm-16.0` and `comm-16.1` have more objects than `comm-16` but less than `comm-17`. For CFF, we did not observe improvements using the modified problems.

We also experimented with using the preprocessor to perform goal splitting and produce modified PDDL files that can be processed by other planners. For example, CFF is unable to solve the problems from p21 to p30 of the `coins` domain. The difficulty in this domain lies in the large number of elevators and coins. The goal-splitting technique divides the problem into a sequence of sub-problems, each dealing with one coin but still has all elevators, enabling CFF to solve these problems. We observed that CFF spent most of the time finding the solution for the first problem. This is reasonable, since the location of the elevators is initially unknown, and some locations will be known at the end of the first solution. The planning time of CFF for coins p21, p25 and p30 is 24.48 2.13 and 68.09 (secs) accordingly.

These initial experiments show that the proposed techniques could be useful for other planners.

7 Conclusions

In this paper, we presented the complete design and implementation of an efficient conformant planner, called CPA(H). The planner builds on recently developed techniques for conformant planning using approximations; it introduces several novelties, including a preprocessing module to transform the problem specification, leading to significantly reduced search spaces, and the ability to explore the search space with different

Problem	t_S	CPA(H) $h_{css(\Sigma)}$	t0	CFF	POND
block-03	0.13	0.22/41	/NA	/AB	1.02/26
block-04	0.18	73.19/312	/NA	/AB	1379/111
coins-10	0.18	0.14/81	0.09/26	1.02/38	5.26/28
coins-15	0.49	5.92/329	0.26/81	7.35/79	/TO
coins-20	0.66	20.24/481	0.32/108	38.19/143	/TO
comm-15	3.64	0.12/95	0.19/110	0.22/95	1662/110
comm-20	141	0.98/239	0.86/278	13.33/239	/ TO
comm-25	1081	3.25/389	3.99/453	109.49/389	/ TO
sortnet-10	0.24	4.21/39	NA	NA	/TO
sortnet-15	0.52	313.04/65	NA	NA	/TO
adder-01	8.2	1.29/3	/NA	/AB	/AB
UTS-cycle-03	0.17	1.08/3	/NA	/NA	1.99/3
UTS-cycle-04	0.29	23.88/6	/NA	/NA	48.19/6
forest-02	2.4	23/84	0.37/12	0.03/17	1.33/15
forest-04	7.7	/TO	1.58/60	/TO	71.17/62
Rao-keys-02	0.22	0.04/32	/NA	0.08/33	0.29/21
Rao-keys-03	0.37	3.84/152	/NA	25.09/101	4.51/68
dispose-8-1	47.3	1.86/137	27.85/291	423.25/226	/AB
dispose-8-2	53.6	391/1328	118.46/422	/AB	/AB
dispose-10-1	285	7.69/213	275.08/474	/AB	/AB

Table 4. Comparison between systems (NA: not applicable)

Domain/BM	of instances	CPA(H) $h_{css(\Sigma)}$	t0	CFF	POND
block/I6	4	3	0	1	3
adder/I5	4	1	0	0	0
coins/I5	30	20	20	20	10
comm/I5	25	25	25	25	16
sortnet/I5	15	15	0	0	6
uts/I5	30	30	30	30	24
UTS-cycle/I6	27	2	0	0	2
forest/I6	9	1	8	1	2
Rao-keys/I6	29	2	0	2	2
dispose/C	90	62	50	41	8
push/C	90	29	33	27	12
1-dispose/C	90	24	7	1	8
look-n-grab/C	81	63	20	21	9

Table 5. Number of problems solved in different domains

heuristic functions. The result is a conformant planner that has been shown to be highly competitive with the state-of-the-art in the field. In particular, CPA(H) outperforms all existing systems on the problems from the latest International Planning Competition.

The results presented in this paper confirm the strength of using approximations for conformant planning, the possibility of implementing approximation-based planning in an efficient and scalable system, and the scope for improvement that can be achieved via transformation of problem specifications.

Problem	Orig/Modified	Problem	Orig/Modified
comm-15	1662/4.89	coins-5	0.52/0.51
comm-16	TO/57.93	coins-10	5.54/1.63
comm-16.0	TO/124.05	coins-15	17.13/17.95
comm-16.1	TO/267.39	coins-10	143/126

Table 6. Impact of `oneof`-Combination on POND (Orig/ Modified: Time for solving the original/modified problem)

The future developments of this project include exploring whether alternative methods for the internal implementation of cs-states (e.g., OBDD) can further enhance performance. We also plan to expand the reasoning component of the preprocessor, to obtain additional simplifications of the problem specifications (e.g., detecting symmetries between fluents).

Acknowledgement

The authors are partially supported by the NSF grants IIS-0812267, CBET-0754525, CNS-0220590, and CREST-0420407.

References

1. C. Baral et al. Computational complexity of planning and approximate planning in the presence of incompleteness. *AIJ*, 122:241–267, 2000.
2. P. Bertoli et al. Heuristic search + symbolic model checking efficient conformant planning. *IJCAI*, pages 467–472, 2001.
3. B. Bonet and B. Givan. Results of the conformant track of the 5th planning competition, 2006. `http://www.ldc.usb.ve/ bonet/`.
4. R. Brafman and J. Hoffmann. Conformant planning via heuristic forward search: A new approach. *ICAPS-04*, pages 355–364, 2004.
5. D. Bryce and S. Kambhampati. Heuristic Guidance Measures for Conformant Planning. *ICAPS-04*, pages 365–375, 2004.
6. D. Bryce et al. Planning Graph Heuristics for Belief Space Search. *JAIR*, 26:35–99, 2006.
7. A. Cimatti et al. Conformant Planning via Symbolic Model Checking and Heuristic Search. *Artificial Intelligence Journal*, 159:127–206, 2004.
8. J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14:253–302, 2001.
9. J. Hoffmann et al. Ordered landmarks in planning. *JAIR*, 22:215–278, 2004.
10. D. Long and M. Fox. . Efficient Implementation of the Plan Graph in STAN. *JAIR*, 10, 1999.
11. X.L Nguyen et al. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *AIJ*, 135:73–123, 2002.
12. H. Palacios and H. Geffner. Compiling Uncertainty Away: Solving Conformant Planning Problems Using a Classical Planner. *AAAI*, 2006.
13. H. Palacios and H. Geffner. From Conformant into Classical Planning: Efficient Translations that may be Complete Too. *ICAPS-07*, 2007.
14. D.E. Smith and D.S. Weld. Conformant graphplan. *AAAI*, pages 889–896, 1998.
15. T.C. Son and C. Baral. Formalizing sensing actions - a transition function based approach. *Artificial Intelligence*, 125(1-2):19–91, January 2001.
16. T.C. Son and P.H. Tu. On the Completeness of Approximation Based Reasoning and Planning in Action Theories with Incomplete Information. *KRR*, 2006.
17. T.C. Son, P.H. Tu, M. Gelfond, and R. Morales. Conformant Planning for Domains with Constraints. *AAAI*, pages 1211–1216, 2005.