Improvements to SAT-based conformant planning*

Claudio Castellini[1,2], Enrico Giunchiglia[2], and Armando Tacchella[3]

[1] Division of Informatics — Univ. of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, UK
[2] DIST — Università di Genova, Viale Causa 13, 16145, Genova, Italia
[3] Dept. of Computer Science — Rice University, 6100 Main St. MS132, 77005 Houston, Texas

Abstract. Planning as satisfiability is an efficient technique for classical planning. In previous work by the second author, this approach has been extended to conformant planning, that is, to planning domains having incomplete information about the initial state and/or the effects of actions. In this paper we present some domain independent optimizations to the basic procedure described in the previous work. A comparative experimental analysis shows that the resulting procedure is competitive with other state-of-the-art conformant planners on domains with a high degree of parallelism.

1 Introduction

Planning as satisfiability [1] is an efficient technique for classical planning. In the classical setting, the idea behind planning as satisfiability is simple: For any action description D, it is possible to compute a propositional formula tr_i^D whose satisfying assignments correspond to the possible transitions caused by the execution of an action in D. In tr_i^D there is a propositional variable A_i for each ground action symbol A, and two propositional variables F_i and F_{i+1} for each ground fluent F in D. Intuitively, F_i represents the value of F in the initial state of the transition, and F_{i+1} represents the value of F in the resulting state, after having performed the action. Then, the problem of finding a plan of length n leading from an initial state I to a goal state G corresponds to finding an assignment satisfying

$$I_0 \wedge \wedge_{i=0}^{n-1} tr_i^D \wedge G_n \tag{1}$$

(see [2] for more details). This simple idea had a lot of impact in the classical planning community, mainly because it led to very impressive results (see, e.g., [2]). In [3], the planning as satisfiability approach has been extended to conformant planning, that is, to planning problems having incomplete information about the initial state and/or the effects of actions. Some preliminary experimental results reported in [4] show that the approach can be competitive w.r.t. CGP [5], a conformant planner based on plan graphs [6].

* A special thank to Paolo Ferraris for many fruitful discussions on the topic of this paper. Paolo has also participated to the design of the architecture of $\mathcal{C}$-PLAN, and has developed the first version of $\mathcal{C}$-PLAN. Norman McCain has made possible to integrate the Causal Calculator in $\mathcal{C}$-PLAN. The first two authors are supported by ASI, CNR and MURST. The third author is partially supported by NSF grants CCR-9700061 and CCR-9988322, BSF grant 9800096, and a grant from the Intel Corporation.

In this paper we present some domain independent optimizations to the basic procedure described in [3]. Some of these optimizations have been incorporated in $\mathcal{C}$-PLAN, a SAT-based system for planning in domains whose action description component is specified using the highly expressive action language $\mathcal{C}$ [7]. As a consequence, $\mathcal{C}$-PLAN allows for conformant planning in domains with constraints, concurrent actions, and nondeterminism. A comparative experimental analysis shows that $\mathcal{C}$-PLAN is competitive with other state-of-the-art conformant planners on domains with an high degree of parallelism.

The paper is structured as follows. In Section 2 we briefly review the conformant planning via SAT approach. In Section 3 we discuss the weaknesses_of and optimizations_to the basic procedure. In Section 4 we perform the experimental comparative analysis. We end the paper in Section 5 with some conclusions.

The material here presented is part of [8]. In [8], we give the precise definitions, statements, proofs, and we also present some additional experimental analysis.

2 Conformant planning via SAT

This Section introduces some terminology and notation that will be used in the rest of the paper. Precise definitions are not given for lack of space. See [3] or [8].

We start with a set of atoms partitioned into a set of *fluent symbols* and a set of *action symbols*. A *formula* is a propositional combination of atoms. An *action* is an interpretation of the action symbols. Intuitively, to execute an action α means to execute concurrently the "elementary actions" represented by the action symbols satisfied by α. In the rest of the paper, an action α is represented by the set of action symbols satisfied by α.

An *action description* D is a finite set of expressions describing how actions change the state of the world, i.e., describing their preconditions and (possibly nondeterministic) effects. Regardless of how D is specified, we assume to have a formula tr_i^D whose satisfying assignments correspond to the transitions of D, as in the classical planning as satisfiability framework outlined in the introduction.[1]

A *planning problem* for D is characterized by two formulas I and G in the fluent signature, i.e., is a triple $\pi = \langle I, D, G\rangle$, where I and G encode the initial and goal state(s) respectively. A *plan* (of *length* $n \geq 0$) is a finite sequence $\alpha^1; \ldots; \alpha^n$ of actions.

Consider a planning problem $\pi = \langle I, D, G\rangle$. A plan $\alpha^1; \ldots; \alpha^n$ is *possible* for π if, starting from an arbitrarily chosen initial state, the consecutive execution of the actions $\alpha^1, \ldots, \alpha^n$ can lead to a goal state. According to the results in [9, 7], possible plans of length n correspond to the assignment satisfying (1), as in the classical setting. However, if we have incomplete information about the initial state and/or actions are non deterministic, then possible plans are not guaranteed to be valid. As pointed out in [9], in order for a plan $\alpha^1; \ldots; \alpha^n$ to be valid we have to check

- that the plan is "always executable" in any initial state, i.e., executable for any initial state and any possible outcome of the actions in the plan, and
- that any "possible result" of executing the plan in any initial state is a goal state.

[1] In [3], we use the term "causally explained transition": here we simply say transition.

$P := I_0 \wedge \wedge_{i=0}^{n-1} tr_i^D \wedge G_n; \qquad V := I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} trt_i^D \wedge \neg(G_n \wedge \neg Z_n);$

function $\mathcal{C}$-SAT() **return** $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$(*cnf*(P), $\{\}$).

function $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$(φ, μ)
  if $\varphi = \{\}$ **then return** $\mathcal{C}$-SAT_TEST(μ);
  if $\{\} \in \varphi$ **then return** *False*;
  if { a unit clause $\{L\}$ occurs in φ } **then return** $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$(*assign*(L, φ),$\mu \cup \{L\}$);
  L := { a literal occurring in φ };
  return $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$(*assign*(L, φ),$\mu \cup \{L\}$) **or** $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$(*assign*$(\overline{L}, \varphi)$,$\mu \cup \{\overline{L}\}$).

function $\mathcal{C}$-SAT_TEST(μ)
  α := { the set of literals in μ corresponding to action literals};
  foreach {plan $\alpha^1; \ldots; \alpha^n$ s.t. each element in α is a conjunct in $\wedge_{i=0}^{n-1} \alpha_i^{i+1}$}
    if not SAT($\wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge V$) **then exit with** $\alpha^1; \ldots; \alpha^n$;
  return *False*.

Fig. 1. $\mathcal{C}$-SAT, $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$ and $\mathcal{C}$-SAT_TEST.

As shown in [3], we can check if a plan $\alpha^1; \ldots; \alpha^n$ is valid for π by verifying whether

$$I_0 \wedge State_0^D \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D \models G_n \wedge \neg Z_n. \tag{2}$$

where

- $State_0^D$ is a formula representing the set of "possible initial states",
- Z is a newly introduced fluent symbol, and
- trt_i^D is a formula defined on the basis of tr_i^D.

Thus, we may divide the problem of finding a valid plan for π into two parts:

1. *generate* possible plans $\alpha^1; \ldots; \alpha^n$ by finding assignments satisfying the formula (1), and
2. *test* whether each generated plan $\alpha^1; \ldots; \alpha^n$ is valid by verifying whether (2) holds.

This is the idea behind the procedure $\mathcal{C}$-SAT in Figure 1. In the Figure, *cnf*(P) is a set of clauses corresponding to P, $\overline{L}$ is the literal complementary to L and, for any literal L and set of clauses φ, *assign*(L, φ) is the set of clauses obtained from φ by deleting the clauses in which L occurs as a disjunct, and eliminating $\overline{L}$ from the others. As it can be seen, $\mathcal{C}$-SAT is the Davis-Logemann-Loveland (DLL) procedure [10], except that, as soon as one assignment satisfying *cnf*(P) is found, the procedure $\mathcal{C}$-SAT_TEST is invoked. Indeed, each assignment satisfying *cnf*(P) corresponds to a set of possible plans, and $\mathcal{C}$-SAT_TEST checks whether any of these are valid. Since all the possible assignments satisfying *cnf*(P) are potentially generated and tested, $\mathcal{C}$-SAT is correct and complete for π, n: Any returned plan is valid for π (*correctness*); and, if *False* is returned, there is no valid plan of length n for π (*completeness*). However, $\mathcal{C}$-SAT only checks the existence of valid plans of length n. Indeed, even assuming that a plan is returned, we are not guaranteed of its optimality (we say that a plan of length n is *optimal* if it is valid and there is no valid plan of length $< n$). Thus, if we are looking for optimal plans, we have to consider $n = 0, 1, 2, 3, 4, \ldots$, and for each value of n, call $\mathcal{C}$-SAT. This is the idea behind the system $\mathcal{C}$-PLAN [4, 8].

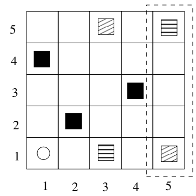

Fig. 2. A robot navigation problem

3 Improvements to SAT-Based conformant planning

In order to understand how the basic procedure above described works consider the following robot navigation problem: We are given a 5×5 grid, and 1 robot is moving in it. It starts from the bottom-left corner of the grid and its goal is to reach the right side. The robot can move $north, east, south, west$, and its duty is not trivial because there can be some objects in some locations of the grid. In order to have some nondeterminism, we assume to have one object in locations $(1,4)$, $(2,2)$, $(4,3)$, and also that either locations $(3,1)$, $(5,5)$ or $(3,5)$, $(5,1)$ are occupied. Figure 2 depicts the resulting scenario. The initial position of the robot is indicated by a circle and occupied locations are marked with a black square. A dashed line outlines the goal locations. Squares filled with a pattern indicate that the corresponding locations *may* be occupied.

According to our definitions in the previous Section, we see that the shortest possible plans are of length 5: Both

$$\{east\}; \{east\}; \{east\}; \{north\}; \{east\}$$

and

$$\{east\}; \{east\}; \{north\}; \{east\}; \{east\}$$

are possible. On the other hand, any optimal valid plan, e.g.,

$$\{north\}; \{north\}; \{east\}; \{east\}; \{south\}; \{east\}; \{east\}$$

is at least 7 steps long. However, before finding a valid plan, $\mathcal{C}$-PLAN will generate and test all the possible plans of length 5, 6, and it may generate also some of the possible plans of length 7. (The possible plans of length 6 are those in which a robot does not move for one time step). Indeed, the main weaknesses of $\mathcal{C}$-PLAN are that, at each n,

1. it generates and tests *all* the possible plans. Indeed, in many cases we can generate only a subset of the possible plans, at the same time keeping completeness of the procedure, and
2. there is no interaction between generation and testing: This may lead to explore huge portions of the search space without finding a solution because of some wrong choices at the beginning of the search tree.

In the following Subsections we describe two domain independent optimizations which alleviate the first two weaknesses.

3.1 Eliminating possible plans

The basic idea is to consider only plans which are possible in a "deterministic version" of the original planning problem. In a deterministic version, all sources of nondeterminism (in the initial state and/or in the effects of the actions) are eliminated. This is possible without losing completeness, and it is a consequence of the following fact.

Consider two planning problems $\pi = \langle I, D, G \rangle$ and $\pi' = \langle I', D', G' \rangle$ in the same fluent and action signatures, and such that

1. every initial state of D' is also an initial state of D, (i.e., $I' \supset I$ is valid),
2. for every action α, the set of states of D in which α is executable is a subset of the set of states of D' in which α is executable,
3. every transition of D' is also a transition of D, (i.e., $tr^{D'} \supset tr^{D}$ is valid),
4. every goal state of π is also a goal state of π' (i.e., $G \supset G'$ is valid).

If a plan is not possible for π' then it is not valid for π. Then, in $\mathcal{C}$-PLAN we can:

- generate possible plans for π', and
- test whether each of the generated possible plans is indeed valid.

The result is still a correct and complete planning procedure (for the given planning problem π and length n). Indeed, in choosing π', we want to minimize the set of possible plans generated and then tested. Hence, we want π' to be a deterministic version of π. A planning problem $\pi' = \langle I', D', G' \rangle$ is a deterministic version of π if it also satisfies the following conditions:

1. I' is satisfied by a single state,
2. for each action α, the set of states in which α is executable in D is equal to the set of states of D' in which α is executable,
3. for any action α and state σ, there is at most one state σ' such that $\langle \sigma, \alpha, \sigma' \rangle$ is a transition of D',
4. G is equal to G'.

Of course, unless the planning problem π is already deterministic, there are many deterministic versions (possibly exponentially many). The obvious question is whether there is one which is "best" according to some criterion. Going back to our robot navigation problem, we have two deterministic versions:

- If locations $(3, 1), (5, 5)$ are occupied, the shortest plan (in the deterministic domain) has length $N_1 = 7$,
- If locations $(3, 5), (5, 1)$ are occupied, the shortest plan has length $N_2 = 5$.

Indeed, any valid plan for the original nondeterministic planning problem has length greater or equal to 7, i.e., to the max between N_1 and N_2. This is not by chance. In fact, let S be the set of deterministic versions of π. For each planning problem π' in S, let $N(\pi')$ be the length of the shortest plan for π'. Then, the length of any valid plan for π is greater or equal to

$$max_{\pi' \in S} N(\pi').$$

On the basis of this fact, we can introduce an order on S according to which $\pi' \in S$ is better than $\pi'' \in S$ if

$$N(\pi') \geq N(\pi''). \tag{3}$$

In other words, we prefer the deterministic versions which start to have solutions (each corresponding to a possible plan for the original planning problem) for the biggest possible value of n. In our robot navigation problem, this would lead us to choose the deterministic version in which the locations $(3,1), (5,5)$ are the ones which are occupied.

Determining the set S of deterministic versions of π is in general not an easy task. Even assuming that determining S is possible, finding the $\pi' \in S$ such to satisfy (3) for each $\pi'' \in S$ seems impractical at best. What we propose is the following. For simplicity, we assume to have nondeterminism only in the initial state: Analogous considerations hold for actions with nondeterministic effects. Under this assumption, we modify our $\mathcal{C}$-SAT_GEN$_{\mathrm{DLL}}$ procedure in Figure 1 in order to do the following:

- once an assignment μ satisfying *cnf*(P) is found, we determine the assignment $\mu' \subseteq \mu$ to the fluent variables at time 0,
- if the possible plan corresponding to μ is not valid, and the planning problem is not already deterministic, then we disallow future assignments extending μ', by adding to I the clause consisting of the complement of the literals satisfied by μ'.

In this way, we progressively eliminate some initial states for which there is a deterministic version having a possible plan of length n. Given that some initial states have not yet been eliminated, the corresponding deterministic versions of the original planning problem have length $\geq n$. At the end, i.e., when we get to a deterministic planning problem π', π' is a deterministic version of π and it satisfies (3) for each $\pi'' \in S$.

In the example of Figure 2, this optimization has the effect of eliminating the initial state in which $\{(3,5),(5,1)\}$ are occupied. This elimination occurs immediately after the first possible but not valid plan (e.g., $\{east\};\{east\};\{east\};\{north\};\{east\}$) is found.

3.2 Incorporating Backjumping and Learning

We do not enter into the details about how backjumping and learning are implemented in SAT, and assume that the reader is familiar with the topic (see, e.g., [11–13]). Here we do the same as in [12, 13], except that we have to extend the procedure there described in order to account for the rejection of assignments corresponding to possible but not valid plans. Indeed, what we can simply do — assuming μ is an assignment corresponding to a possible but not valid plan $\boldsymbol{\alpha} = \alpha^1;\ldots;\alpha^n$ — is to return *False* and set $\vee_{i=0}^{n-1} \neg\alpha_i^{i+1}$ as the initial working reason. However, are there any other (better) choices? According to the definition of valid plan, there are two possible causes why $\boldsymbol{\alpha}$ is not valid:

1. executing $\alpha^1;\ldots;\alpha^k$ in some initial state can lead to a state σ^k, and α^{k+1} is not executable in σ^k, or
2. $\boldsymbol{\alpha}$ is always executable in any initial state, but one of the possible outcomes of executing $\boldsymbol{\alpha}$ in an initial state is not a goal state.

In both cases, $\mathcal{C}$-SAT_TEST determines an assignment μ' satisfying $\wedge_{i=0}^{n-1} \alpha_i^{i+1} \wedge V$, and thus returns *False*. Also notice that in the first case, μ' satisfies

$$\neg Z_0, \ldots, \neg Z_k, Z_{k+1}, \ldots, Z_n$$

$\|P\|$-$\|T\|$	GPT Total	CMBP #s	CMBP Total	Cplan #s	Cplan #pp	Cplan Last	Cplan Tot.search	Cplan Total
2-1	0.03	2	0.00	1	1	0.00	0.00	0.00
4-1	0.03	4	0.01	1	1	0.00	0.00	0.00
6-1	0.04	6	0.02	1	1	0.00	0.00	0.00
8-1	0.15	8	0.08	1	1	0.00	0.00	0.00
10-1	0.27	10	0.61	1	1	0.00	0.00	0.00
15-1	17.05	15	42.47	1	1	0.00	0.00	0.00
20-1	MEM	–	MEM	1	1	0.00	0.00	0.00

Table 1. Bomb in the toilet: Classic version

with $k < n$. Then we can set $\vee_{i=0}^{k} \neg\alpha_i^{i+1}$ as the initial working reason for rejecting μ: Any assignment satisfying $\wedge_{i=0}^{k} \alpha_i^{i+1}$ corresponds to a not valid plan. Of course, setting $\vee_{i=0}^{k} \neg\alpha_i^{i+1}$ as working reason for rejecting μ is better than setting $\vee_{i=0}^{n-1} \neg\alpha_i^{i+1}$: since $k < n$ each disjunct in $\vee_{i=0}^{k} \neg\alpha_i^{i+1}$ is also in $\vee_{i=0}^{n-1} \neg\alpha_i^{i+1}$, and, if $k < n-1$, the viceversa is not true.

In the example of Figure 2, this optimization has the following effect: If the possible plan $\{east\}; \{east\}; \{east\}; \{north\}; \{east\}$ is tried, then it is rejected with a reason which inhibits the further generation of possible plans beginning with $\{east\}; \{east\}$.

4 Experimental Analysis

The ideas presented in Section 2 and the optimizations described in Section 3 have been implemented in $\mathcal{C}$-PLAN. $\mathcal{C}$-PLAN accepts action descriptions specified in $\mathcal{C}$, and thus it naturally allows for, e.g., concurrency, constraints and nondeterminism. Our current version (ver. 2) of $\mathcal{C}$-PLAN has been implemented on top of SIM, an efficient SAT checker developed by our group [13].

To evaluate $\mathcal{C}$-PLAN's effectiveness we consider an elaboration of the traditional "bomb in the toilet" problem from [14]. There is a finite set P of packages and a finite set T of toilets. One of the packages is armed because it contains a bomb. Dunking a package in a toilet disarms the package and is possible only if the package has not been previously dunked. We first consider planning problems with $|P| = 2, 4, 6, 8, 10, 15, 20$, and $|T| = 1$. We compare our system with Bonet's and Geffner's GPT [15], and Cimatti's and Roveri's CMBP [16, 17] planners. These are two among the most recent conformant planners, and according to the results presented in [17], also the most effective to date. We remark that both CMBP and GPT are sequential planners: they try to execute at most one action at each step. Furthermore, CMBP computes all the valid plans, not just one like GPT and $\mathcal{C}$-PLAN. The results for these three systems are shown in Table 1. In the table, we show

- for GPT, the total time the system takes to solve the problem,
- for CMBP, the number of steps (column "#s") (i.e., the number of elementary actions) and the total time needed to solve the problem (column "Total"),
- for $\mathcal{C}$-PLAN,
  - the number of steps (i.e., the number of parallel actions) (column "#s");

- the number of possible plans generated before finding a valid one (column "#pp");
- the search time taken by the system at the last step (column "Last");
- the total search time, being the sum over all steps of the time taken by $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$ and $\mathcal{C}$-SAT_TEST (column "Tot.search"); and
- the total time taken by the system to solve the problem, excluding the off-line time necessary to compute tr_i^D and trt_i^D (column "Total"). This total time does not coincide with "Tot.search" because it includes also the times necessary for expanding the formula, and for doing some other computation internal to the system.

Times are in seconds, and all the tests have been run on a Pentium III, 850MHz, 512MBRAM running Linux SUSE 7.0. For practical reasons, we stopped the execution of a system if its running time exceeded 1200s of CPU time or if it required more than the 512MB of available RAM. In the table, the first case is indicated with "TIME" and the second with "MEM".

As it can be seen from Table 1, $\mathcal{C}$-PLAN takes full advantage of its ability to execute actions concurrently. Indeed, it solves the problem in only one step, by dunking all the packages at the same time. Furthermore, the time taken by $\mathcal{C}$-PLAN is always not measurable. CMBP and GPT have comparable performances, with CMBP being better of a factor of 2-3. However, the most interesting data about these systems is that when $|P| = 20$ they both run out of memory. Indeed, both GPT and CMBP can require huge amounts of memory: GPT for storing the set of belief states visited, and CMBP for storing (as Binary Decision Diagrams, BDD [18]) the transition relation and the set of belief states visited.

We also consider the elaboration of the "bomb in the toilet" in which dunking a package clogs the toilet. There is the additional action of flushing the toilet, which is possible only if the toilet is clogged. The results are shown in Table 2 for $|P| = 2, 4, 6, 8, 10$ and $|T| = 1, 5, 10$. With one toilet, these problems are the "sequential version" of the previous. With multiple toilets they are similar to the "BMTC" problems in [17]. As we can see from Table 2, when there is only one toilet $\mathcal{C}$-PLAN "Total" time increases rapidly compared to the other solvers. Indeed, $|T| = 1$ represents the purely sequential case in which the only valid plan consists in repeatedly dunking a package and flushing the toilet till all the packages have been dunked. On the other hand, we see, e.g., for $|P| = 6$ and $|T| = 1$, that most of $\mathcal{C}$-PLAN time is not spent in the search. By analysing these numbers and profiling $\mathcal{C}$-PLAN code, we discovered that

- most of the search time is spent by $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$: on all the experiments we tried, each call of $\mathcal{C}$-SAT_TEST takes an hardly measurable time,
- the time taken by the system to expand the formula at each step can be considerable, but (since the expansion is done once for each step) does not account for the possible big differences between "Tot.search" and "Total". This is evident if we compare row 6-1 in Table 2 with row 6-1 in Table 3: The formulas to expand in the two cases are equal, and the number of expansion is the same. However, the "Total" time in Table 2 is significantly bigger than the "Total" time in Table 3,
- the possible big differences are due to the fact that, in our current version, each time $\mathcal{C}$-SAT_TEST is invoked by $\mathcal{C}$-SAT_GEN$_{\text{DLL}}$, we pay a cost linear in the size of the V formula. This cost is due to the copying of the V formula from one data-structure

	GPT	CMBP		Cplan				
$\|P\|$-$\|T\|$	Total	#s	Total	#s	#pp	Last	Tot.search	Total
2-1	0.10	3	0.00	3	6	0.00	0.00	0.01
2-5	0.04	2	0.01	1	1	0.00	0.00	0.00
2-10	0.05	2	0.03	1	1	0.00	0.00	0.00
4-1	0.04	7	0.00	7	540	0.12	0.15	0.65
4-5	0.23	4	0.79	1	1	0.00	0.00	0.00
4-10	2.23	4	11.30	1	1	0.00	0.00	0.01
6-1	0.09	11	0.04	11	52561	15.39	49.39	221.55
6-5	3.29	7	16.80	3	98346	56.92	57.34	419.53
6-10	74.15	–	MEM	1	1	0.00	0.00	0.01
8-1	0.41	15	0.20	–	–	–	–	TIME
8-5	32.07	11	112.48	–	–	–	–	TIME
8-10	MEM	–	MEM	1	1	0.00	0.00	0.01
10-1	2.67	19	1.55	–	–	–	–	TIME
10-5	MEM	15	974.45	–	–	–	–	TIME
10-10	MEM	–	MEM	1	1	0.00	0.00	0.04

Table 2. Bomb in the toilet: Multiple toilets, clogging, one bomb.

to another internal of SIM. This linear cost, multiplied by the number of times $\mathcal{C}$-SAT_TEST is invoked, accounts for the difference between "Tot.search" and "Total" in the example 6-1 of Table 2.

We remark that the potentially exponential cost of verifying a plan does not arise in practice, at least on all the experiments we tried. As a matter of fact, each time a plan is verified, the corresponding set of unit clauses is added to the V formula and (if the plan is not valid) the empty set of clauses is generated after very few splits. This was expected (see the Section on implementation in [3]): what we underestimated is the cost paid because of copying the V formula. We believe that this cost is also responsible for many of $\mathcal{C}$-PLAN's timeouts, and that a better engineering of the system, meant to solve this particular problem, will allow $\mathcal{C}$-PLAN to be even more competitive. In any case, $\mathcal{C}$-PLAN's performances are not too bad compared to the ones of the other solvers: $\mathcal{C}$-PLAN, CMBP, GPT do not solve 4, 3, 3 problems respectively.

Finally, we consider the same problem as before, except that we do not know how many packages are armed. These problems, with respect to the ones previously considered, present a higher degree of uncertainty. We consider the same values of $|P|$ and $|T|$ and report the same data as before. The results are shown in Table 3.

Contrarily to what could be expected, $\mathcal{C}$-PLAN performances are much better on the problems in Table 3 than on those in Table 2. This is most evident if we compare the number of plans generated and tested by $\mathcal{C}$-PLAN before finding a solution. For example, if we consider the four packages and one toilet problem,

– with one bomb, as in Table 2, $\mathcal{C}$-PLAN generates 540 possible plans and takes 0.65s to solve the problem (0.15s of search time),
– with possibly multiple bombs, as in Table 3, $\mathcal{C}$-PLAN generates 15 possible plans and takes 0.02s to solve the problem (0.02s is also the search time).

$\|P\|$-$\|T\|$	GPT Total	CMBP #s	CMBP Total	Cplan #s	Cplan #pp	Cplan Last	Cplan Tot.search	Cplan Total
2-1	0.03	3	0.00	3	3	0.00	0.00	0.00
2-5	0.04	2	0.00	1	1	0.00	0.00	0.00
2-10	0.24	2	0.02	1	1	0.00	0.00	0.02
4-1	0.17	7	0.01	7	15	0.01	0.02	0.02
4-5	0.06	4	0.54	1	1	0.01	0.00	0.01
4-10	0.38	4	7.13	1	1	0.02	0.00	0.02
6-1	0.08	11	0.03	11	117	0.25	1.39	2.01
6-5	0.33	7	10.71	3	48	0.62	0.66	1.36
6-10	7.14	–	MEM	1	1	0.00	0.00	0.00
8-1	0.06	15	0.17	15	1195	12.23	147.25	184.29
8-5	2.02	11	90.57	3	2681	14.84	15.60	317.13
8-10	MEM	–	MEM	1	1	0.00	0.00	12.68
10-1	0.21	19	1.02	–	–	–	–	TIME
10-5	12.51	15	591.33	–	–	–	–	TIME
10-10	MEM	–	MEM	1	1	0.00	0.00	0.06

Table 3. Bomb in the toilet: Multiple toilets, clogging, possibly multiple bombs.

To understand why, consider for simplicity the case in which there is only one toilet and two packages P_1 and P_2. For $n = 0$

- there are no possible plans if we know that there is one bomb, and
- there is the possible plan consisting of the empty sequence of actions, corresponding to assuming that neither P_1 nor P_2 is armed, in the case we know nothing. This plan is not valid, and, because of the determinization, $\mathcal{C}$-PLAN adds a clause to the initial state saying that at least one package is armed.

For $n = 1$, $\mathcal{C}$-PLAN in both cases tries 2 possible plans. If we assume that it generates first the plan in which P_1 is dunked

- if we further assume that there is one bomb, it rejects it, and —because of the determinization— it adds a clause to the initial state which allows to conclude that the bomb is in P_2. Then, for $n = 2$ and $n = 3$, any plan in which P_2 is dunked is possible.
- in the other case, it rejects it, and —because of the determinization— it adds a clause to the initial state saying that the bomb is not in P_1 or is in P_2. Then, it generates the other plan in which only P_2 is dunked. Also this plan is rejected and a clause saying that a bomb is in P_1 or not in P_2 is added to the initial state. Thus, there is now only one initial state satisfying all the constraints: namely the one in which both P_1 and P_2 are armed. This allows $\mathcal{C}$-PLAN to conclude that there are no possible plans for $n = 2$, and to immediately generate a valid plan at $n = 3$.

The optimizations described in Section 3 help a lot. Indeed, if we consider the four packages and one toilet problem, and disable the optimizations,

- if we have one bomb, as in Table 2, $\mathcal{C}$-PLAN generates 2145 possible plans and takes 0.54s to solve the problem (0.24s in the last step),

– if we have possibly multiple bombs, as in Table 3, $\mathcal{C}$-PLAN generates 3743 possible plans and takes 0.93s to solve the problem (0.72s in the last step).

Besides $\mathcal{C}$-PLAN's performances, also CMBP and GPT seem to get benefits by the added nondeterminism. Overall, $\mathcal{C}$-PLAN, CMBP and GPT do not solve respectively 2, 3 and 2 of the considered problems. On the ones they solve, we get roughly the same picture that we had before: $\mathcal{C}$-PLAN takes full advantage of its ability to concurrently execute actions, and thus behaves better on problems with multiple toilets.

Overall, GPT, CMBP and $\mathcal{C}$-PLAN do not solve 6, 7, 6 respectively of the 37 examples that we tried.

5 Conclusions and related work

We have presented some optimizations to the basic procedure for conformant planning described in [3]. The procedure and the optimizations have been implemented in $\mathcal{C}$-PLAN ver. 2, a SAT-based conformant planner based on the SIM SAT library. $\mathcal{C}$-PLAN incorporates the Causal Calculator [9], and is thus able to reason about action descriptions specified in $\mathcal{C}$. $\mathcal{C}$ allows for, e.g., concurrency, constraints, and nondeterminism. This causes $\mathcal{C}$-PLAN to be one of most expressive conformant planners among the currently available. From the experimental analysis we get that GPT, CMBP and $\mathcal{C}$-PLAN do not solve 6, 7, 6 respectively of the 37 examples that we tried. Most important, while $\mathcal{C}$-PLAN runs out of time, CMBP and GPT run out of memory. This seems to point out that $\mathcal{C}$-PLAN range of applicability is different from the range of applicability of CMBP and GPT. Analogous results supporting this fact are reported in [19] where it is shown that for classical, highly parallel domains the planning as satisfiability approaches appear to do best. The fact that SAT-based approaches and BDD-based approaches have different range of applicability is also confirmed by

– previous work comparing BDD and DLL as SAT procedures, see [20],
– recent work in symbolic reachability in formal verification, see [21, 22].

Beside the already cited [15, 16], two other works on conformant planning are [5] and [24]. In [5], the authors propose an approach based on plan graphs. The underlying idea is to construct a planning graph for every possible deterministic version of the original planning problem. Constraints over planning graphs ensure conformance. The main weakness of the approach is that there can be exponentially many deterministic version, causing the creation of exponentially many planning graphs. In [24], Rintanen reduces the problem of conformant and conditional planning to the problem of deciding the satisfiability of a Quantified Boolean Formula (QBF). Our approach is similar to Rintanen's: The search performed by our generate and test procedure resembles the one of a QBF solver if run on formulas corresponding to conformant planning problems. On the other hand, by dealing with the original planning problem, we are able to introduce optimizations —like the ones described in Section 3— which take into account the nature of the original problem.

In [23], a new algorithm for conformant planning based on "heuristic-symbolic search", is proposed and the experimental results are impressive. A detailed analysis of the paper and the results is in our agenda.

References

1. Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363.
2. Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic and stochastic search. In *Proc. AAAI-96*, pages 1194–1201.
3. Enrico Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Proc. KR-2000*.
4. Paolo Ferraris and Enrico Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. AAAI-2000*.
5. David Smith and Daniel Weld. Conformant graphplan. In *Proc. AAAI-98*, pages 889–896.
6. Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proc. of IJCAI-95*, pages 1636–1642, 1995.
7. Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630.
8. Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. SAT-based planning in complex domains: Concurrency, constraints and nondeterminism, 2001. Unpublished.
9. Norman McCain and Hudson Turner. Fast satisfiability planning with causal theories. In *Proc. KR-98*.
10. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.
11. Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.
12. Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. AAAI-97*, pages 203–208.
13. Enrico Giunchiglia, Marco Maratea, Armando Tacchella, and Davide Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001)*, 2001.
14. Drew McDermott. A critique of pure reason. *Computational Intelligence*, 3:151–160, 1987.
15. Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. AIPS*, 2000.
16. Alessandro Cimatti and Marco Roveri. Conformant planning via model checking. In *Proc. ECP-99*.
17. Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.
18. Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.
19. Patrick Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *Proc. AIPS-2000*, pages 140–149.
20. T. E. Uribe and M. E. Stickel. Ordered Binary Decision Diagrams and the Davis-Putnam Procedure. In *Proc. of the 1st International Conference on Constraints in Computational Logics*, 1994.
21. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. TACAS-99*.
22. Fady Copty, Limor Fix, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Vardi. Benefits of bounded model checking at an industrial setting. In *Proc. CAV-2001*.
23. A. Cimatti, E. Giunchiglia, M. Pistore, E. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: BDD-based + SAT-based symbolic model checking, 2001. Unpublished.
24. Jussi Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.