SAT-based planning with minimal-#actions plans and "soft" goals

Enrico Giunchiglia and Marco Maratea

DIST, University of Genova, Viale F. Causa 15, Genova, Italy
{enrico,marco}@dist.unige.it

Abstract. Planning as Satisfiability (SAT) is the best approach for optimally solving classical planning problems. The SAT-based planner SATPLAN has been the winner in the deterministic track for optimal planners in the 4th International Planning Competition (IPC-4) and the co-winner in the last 5th IPC (together with another SAT-based planner). Given a planning problem Π, SATPLAN works by (i) generating a SAT formula Π_n with a fixed "makespan" n, and (ii) checking Π_n for satisfiability. The algorithm stops if Π_n is satisfiable, and thus a plan has been found, otherwise n is increased.
Despite its efficiency, and the optimality of the makespan, SATPLAN has significant deficiency related in particular to "plan quality", e.g., the number of actions in the returned plan, and the possibility to express and reason on "soft" goals.
In this paper, we present SATPLAN$^{\prec}$, a system, modification of SATPLAN, which makes a significant step towards the elimination of SATPLAN's limitations. Given the optimal makespan, SATPLAN$^{\prec}$ returns plans with minimal number of actions and maximal number of satisfied "soft" goals, with respect to both cardinality and subset inclusions. We selected several benchmarks from different domains from all the IPCs: on these benchmarks we show that the plan quality returned by SATPLAN$^{\prec}$ is often significantly higher than the one returned by SATPLAN.
Quite surprisingly, this is often achieved without sacrificing efficiency while obtaining results that are competitive with the winning system of the "SimplePreferences" domain in the satisfying track of the last IPC.

1 Introduction

Planning as Satisfiability (SAT) [1] is the best approach for optimally solving classical planning problems. The SAT-based planner SATPLAN [2, 3] has been the winner in the deterministic track for optimal planners in the 4th International Planning Competition (IPC-4) [4] and the co-winner in the recent IPC-5 (together with another SAT-based planner, MAXPLAN [5]). Given a planning problem Π, SATPLAN works by (i) generating a SAT formula Π_n with a fixed "makespan" n, and (ii) checking Π_n for satisfiability. The algorithm stops if Π_n is satisfiable, and thus the plan has been found, otherwise n is increased.

Despite its efficiency, and the optimality of the makespan, SATPLAN has significant deficiency related in particular to "plan quality", e.g., the number of actions in the returned plan, and the possibility to express and reason on "soft" goals. The issues are related with the following facts: in SATPLAN, when solving the propositional formula

Π_n, there is no indication of what propositional variables correspond to actions, and the SAT solver does not perform any kind of optimization on the number of actions in the plan, treating each propositional variable in the same way, independently from what it indicates. The makespan is fixed, but multiple, mutually exclusive (mutex) actions can take place simultaneously, even if often not all actions are relevant to reach the goal. On the other hand, "soft" goals arise in planning problems when there is no possibility to satisfy simultaneously all the goals, and/or when it is sufficient (from the view point of the user) that only some of them are satisfied.

In this paper, we present SATPLAN$^{\prec}$, a system, modification of SATPLAN [3], which makes a significant step towards the elimination of SATPLAN's limitations. Given the optimal makespan, SATPLAN$^{\prec}$ returns plans with the minimal number of actions and maximal number of the satisfied "soft" goals, with respect to both cardinality and subset inclusions. This is achieved by integrating the OPTSAT solver in SATPLAN. OPTSAT [6, 7] is a tool for solving SAT related optimization problems based on the state-of-the-art SAT solver MINISAT. Besides other features, given a SAT formula Π_n and a subset S of the variables in Π_n, OPTSAT returns an "optimal" solution, i.e., a satisfying assignment that minimize/maximize the atom in S assigned to TRUE.

We selected several domains of benchmarks from all the IPCs: on these benchmarks we show that the plan quality returned by SATPLAN$^{\prec}$ is often significantly higher than the one of SATPLAN using both SIEGE and MINISAT. In particular

- SATPLAN$^{\prec}$ is usually able to satisfy a number of soft goals which is much higher than the one of SATPLAN, e.g., there are instances with several soft goals where SATPLAN$^{\prec}$ satisfy all (or almost all) soft goals while SATPLAN is able to satisfy just a few of them.
- SATPLAN$^{\prec}$ usually returns plans with fewer number of actions than SATPLAN.

Quite surprisingly, this results are often achieved without sacrificing efficiency while the obtained results are competitive with the winning system of the "SimplePreferences" domain in the satisfying track of the last IPC, i.e., SGPLAN [8], as shown in [11].

Moreover, a closer look at the performance in terms of metrics for which SATPLAN$^{\prec}$ is optimized, i.e., makespan and number of actions in the plan, in comparison to both SATPLAN and SGPLAN, on benchmarks where each of the solvers satisfies all the soft goals, reveals that (i) SATPLAN and SATPLAN$^{\prec}$ often return plans with a much better makespan than SGPLAN; (ii) on some benchmarks the reduction in terms of number of actions in the plan returned by SATPLAN$^{\prec}$ is very significant; but also that (iii) there are particular instances in which SGPLAN returns plans with fewer actions. Because this is due to the non-optimal makespan returned, (iii) suggests that is could be useful, in order to further improve SATPLAN$^{\prec}$'s performance, to "trade-off" between optimality of the makespan and optimality of the plan quality.

The paper is structured as follows. In Sec 2 some basic preliminaries about planning (as satisfiability) are presented. Sec. 3 is devoted to the details on how the new features of SATPLAN$^{\prec}$ are implemented. In Sec. 4 is then shown how to use SATPLAN$^{\prec}$, its command line and options. Sec. 5 shows the results we have obtained with SATPLAN$^{\prec}$, and finally Sec. 6 draws some conclusions and possible topics for future research.

2 Preliminaries

Let $\mathcal{F}$ and $\mathcal{A}$ be the set of *fluents* and *actions*, respectively. A *state* is an interpretation of the fluent signature. A *complex action* is an interpretation of the action signature. Intuitively, a complex action α models the concurrent execution of the actions satisfied by α.

A *planning problem* is a triple $\langle I, tr, G\rangle$ where

- I is a Boolean formula over $\mathcal{F}$ and represents the set of *initial states*;
- tr is a Boolean formula over $\mathcal{F} \cup \mathcal{A} \cup \mathcal{F}'$ where $\mathcal{F}' = \{f' : f \in \mathcal{F}\}$ is a copy of the fluent signature and represents the *transition relation* of the automaton describing how (complex) actions affect states (we assume $\mathcal{F} \cap \mathcal{F}' = \emptyset$);
- G is a Boolean formula over $\mathcal{F}$ and represents the set of *goal states*.

The above definition of planning problem differs from the traditional ones in which the description of actions' effects on a state is described in an high-level action language like STRIPS or PDDL. We preferred this formulation because the techniques we are going to describe are largely independent of the action language used, at least from a theoretical point of view. The only assumption that we make is that the description is deterministic: there is only one state satisfying I and the execution of a (complex) action α in a state s can lead to at most one state s'. More formally, for each state s and complex action α there is at most one interpretation extending $s \cup \alpha$ and satisfying tr.

Consider a planning problem $\Pi = \langle I, tr, G\rangle$. In the following, for any integer i

- if F is a formula in the fluent signature, F_i is obtained from F by substituting each $f \in \mathcal{F}$ with f_i,
- tr_i is the formula obtained from tr by substituting each symbol $p \in \mathcal{F} \cup \mathcal{A}$ with p_{i-1} and each $f \in \mathcal{F}'$ with f_i.

If n is an integer, the *planning problem Π with makespan n* is the Boolean formula Π_n defined as

$$I_0 \wedge \wedge_{i=1}^{n} tr_i \wedge G_n \qquad (n \geq 0) \tag{1}$$

and a *plan for Π_n* is an interpretation satisfying (1).

For example, considering the planning problem of going to work from home. Assuming that we can use the car or the bus or the bike, this scenario can be easily formalized using a single fluent variable *AtWork* and three action variables *Car*, *Bus* and *Bike* with the obvious meaning. The problem with makespan 1 can be expressed by the conjunction of the formulas:

$$\begin{gathered} \neg \mathit{AtWork}_0, \\ \mathit{AtWork}_1 \equiv \neg \mathit{AtWork}_0 \equiv (\mathit{Car}_0 \vee \mathit{Bus}_0 \vee \mathit{Bike}_0), \\ \mathit{AtWork}_1, \end{gathered} \tag{2}$$

in which the first formula corresponds to the initial state, the second to the transition relation, and the third to the goal state. (2) has 7 plans (i.e., satisfying interpretations), each corresponding to a non-empty subset of $\{\mathit{Car}_0, \mathit{Bus}_0, \mathit{Bike}_0\}$. For instance, in the plan corresponding to $\{\mathit{Car}_0, \mathit{Bus}_0\}$ both the car and the bike are used to get to work.

If we want to avoid any two actions in $\{Car_0, Bus_0, Bike_0\}$ to occur in parallel, the following mutex axioms are to be added as part of the formulas encoding the transition relation

$$\neg(Car_0 \wedge Bus_0), \neg(Car_0 \wedge Bike_0), \neg(Bus_0 \wedge Bike_0).$$

3 SATPLAN$^{\prec}$ implementation

SATPLAN$^{\prec}$ is a modification of the SATPLAN system. When constructing the SAT formula Π_n, a number of information are added to the formula (which in SATPLAN is specified in CNF format), in order to cope with the specific problem. If minimization on the number of actions in the Π is considered, the total number of actions and the list of the propositional variables corresponding to actions are specified in the comment lines of (the CNF file related to) Π_n. The comment lines also specify if the minimization has to be computed under cardinality or subset inclusion.

Otherwise, if we are dealing with maximization of the number of "soft" goals to be satisfied, some of the clauses in Π_n have to be modified, resulting in a new formula Π'_n. Each clause $C \in \Pi_n$ that represents the i-th goal is modified by adding a "goal selectors" s_i, i.e., by substituting C with $C' := \overline{s}_i \cup C$. Intuitively, the clause selectors tell us how many soft goals are satisfied, i.e., assuming C' is satisfied, if s_i is assigned by TRUE this means that clause C' is satisfied by C, thus the related soft goal holds; otherwise, if s_i is assigned by FALSE, this means that it could be the case that C' is only satisfied by the clause selector. For the soft goals, in the command line it is only needed to specify the problem, and the type of minimality.

Now we explain how optimality is obtained. Consider the set of atoms S to be the set that should be optimized, i.e., the set of actions in Π_n or the set of the goal selectors. If the optimality is to be performed under subset inclusion, it suffices to "preferentially" splits on the atom in S, and assign it to FALSE (resp. TRUE). The resulting satisfying assignment μ of Π_n (resp. Π'_n) is guaranteed to contain the minimal number of atoms assigned by TRUE (resp. the maximal number of goal selectors assigned by TRUE), thus resulting in a plan with the minimal number of actions (resp. the maximal number of soft goals satisfied).

On the other hand, if the minimality is by cardinality, we have to compute an auxiliary boolean encoding $Bool(S)$ of S whose implementation depends on whether we rely on (a) a binary or (b) a unary encoding. For any satisfying assignment μ of Π_n, there exists a unique interpretation μ' to the variables in $\Pi_n \wedge Bool(S)$ such that μ' extends μ and satisfies $\Pi_n \wedge Bool(S)$. Given an atom p, consider that $\mu(p)$ is 1 if μ assigns p to true, and is 0 otherwise. $Bool(S)$ contains k new variables $b_{k-1}, \ldots, b_0$ such that

(a) if $k = \lceil log_2(|S| + 1) \rceil$, $\sum_{p \in S} \mu(p) = \sum_{i=0}^{k-1} \mu(b_i) \times 2^i$, or
(b) if $k = |S|$, $\sum_{p \in S} \mu(p) = \sum_{i=0}^{k-1} \mu(b_i)$.

Intuitively, the goal of points (a) and (b) is to encode an optimization function (in our case related to the cardinality $|S|$ of the atoms p in S assigned to TRUE) as a boolean formula $Bool(S)$ which contains a set of atoms $b_{k-1}, \ldots, b_0$ that "characterize" the optimization function.

We considered Warners' [9] and Bailleux and Boufkhad's [10] encodings of *Bool*(S), denoted with W-encoding and B-encoding respectively, as representative encodings for (a) and (b), respectively.

1. Warners. It uses a binary representation of integers. This is a linear time and space encoding, that relies on sums via adder circuits and works directly with objective functions with weights. In OPTSAT, and thus in SATPLAN$^{\prec}$, the encoding is optimized for the non-weighted case, and the size of the encoding is approximately halved.

   Example 1. Consider $S = \{p_1, p_2, p_3\}$. *Bool*(S) is the set of clauses corresponding to the sum of three variables, i.e., $\{\{p_7, p_4, \neg p_1\}, \{p_7, \neg p_4, p_1\}, \{\neg p_7, \neg p_4, \neg p_1\}, \{\neg p_7, p_4, p_1\}, \{p_6, \neg p_4, \neg p_1\}, \{\neg p_6, p_4\}, \{\neg p_6, p_1\}, \{p_8, p_5, \neg p_6\}, \{p_8, \neg p_5, p_6\}, \{\neg p_8, \neg p_5, \neg p_6\}, \{\neg p_8, p_5, p_6\}, \{p_4, p_2, \neg p_3\}, \{p_4, \neg p_2, p_3\}, \{\neg p_4, \neg p_2, \neg p_3\}, \{\neg p_4, p_2, p_3\}, \{p_5, \neg p_2, \neg p_3\}, \{\neg p_5, p_2\}, \{\neg p_5, p_3\}\}$.

   The b_i variables are p_7 and p_8, and this corresponds to $\sum_{i=1}^{3} \mu(p_i) = 2^1\mu(p_8) + 2^0\mu(p_7)$, while p_4, p_5 and p_6 are added by the encoding.

2. Bailleux/Boufkhad (B). In this encoding a unary representation of integers is used: an integer x s.t. $0 \leq x \leq z$ is represented using z propositional variables $\{p_1, \ldots, p_z\}$ with (the first) "x" variables assigned to 1 (TRUE), and the others to 0 (FALSE). This representation has the property that when a variable p_j has value TRUE, all the variables p_w with $1 \leq w < j$, are TRUE as well; and similarly if p_k has value FALSE. The encoding is efficient with respect to unit-propagation but it adds a quadratic number of new clauses.

   Example 2. Consider the same Example used in 1, *Bool*(S) is now $\{\{\neg p_6, p_1, p_4\}, \{p_6, \neg p_4\}, \{\neg p_7, p_1, p_5\}, \{p_7, \neg p_5\}, \{\neg p_8, p_1\}, \{p_6, \neg p_1\}, \{\neg p_7, p_4\}, \{p_7, \neg p_1, \neg p_4\}, \{\neg p_8, p_5\}, \{p_8, \neg p_1, \neg p_5\}, \{\neg p_4, p_2, p_3\}, \{p_4, \neg p_3\}, \{\neg p_5, p_2\}, \{p_4, \neg p_2\}, \{\neg p_5, p_3\}, \{p_5, \neg p_2, \neg p_3\}\}$, in which the b_i variables are p_6, p_7 and p_8, $\sum_{i=1}^{3} \mu(p_i) = \mu(p_6) + \mu(p_7) + \mu(p_8)$, with p_8 being the most significant variable (i.e., b_2), while p_4 and p_5 are introduced by the encoding.

We also exploited a further alternative, modification of the B-encoding, leveraging on its representation. We noticed that the encoding does not take into full account the relations among the resulting b_i variables, e.g., it is safe to enforce that when b_i is assigned to FALSE (resp. TRUE), also b_{i+1} (resp. b_{i-1}) is assigned to FALSE (resp. TRUE) by unit propagation. This is easily done by adding the clauses $\{b_i, \neg b_{i+1}\}$, $i = 0, \ldots, v-1$ to the above encoding. Given that this modification did not show significant enhancement over B-encoding, we will not consider it in the design of SATPLAN$^{\prec}$ and in the experimental evaluation.

Despite the difference in the size of the encodings, the B-encoding has better computational properties and it has been consistently reported in the literature to lead to good results [10, 12].

Given $b_{k-1}, \ldots, b_0$, preferentially and in-order splitting from b_{k-1} to b_0 and assign it with the "needed" value (i.e., the one resulting from the combination of optimization and minimality, following the schema of assignment we used when dealing with a subset inclusion minimality) leads to an "optimal" solution (see [7] for more details).

As we noticed before, this is achieved by using the OPTSAT system, instead of invoking a basic SAT solver, like MINISAT or SIEGE.

A final, important consideration has to be made. We have seen that in the last IPC-5 SATPLAN was the winner together with MAXPLAN. The question is whether the new features we have proposed can be simply integrated into MAXPLAN. MAXPLAN works by firstly estimating an upper bound n of the optimal makespan, and than (i) generating a SAT formula Π_n for the fixed makespan n, and (ii) checking Π_n for satisfiability, by using a modified version of MINISAT. The algorithm stops if Π_n is unsatisfiable, otherwise n is decreased. Given this, it should be relatively easy to integrate the new features of SATPLAN$^{\prec}$ into MAXPLAN.

4 SATPLAN$^{\prec}$ command line and options

SATPLAN$^{\prec}$ has to be invoked as follows:

./SATPLAN$^{\prec}$ -solver <> -opt <> -minimality <> -enc <> -problem <> -domain <>

The command line of SATPLAN$^{\prec}$ is thus very similar to the one of SATPLAN. Indeed, SATPLAN$^{\prec}$ accepts all the options of SATPLAN (here we report only the mandatories "-problem" and "-domain"), and adds new options in order to deal with the new features of SATPLAN$^{\prec}$.

Going in more details,

-solver: indicates the solver to be used for solving the SAT problem. Among others, the most important are OPTSAT, MINISAT and SIEGE ("optsat1.0", "minisat1.14" and "siege"are the specific strings to be specified in place of <>). If the goal is to use the new features of SATPLAN$^{\prec}$, it is mandatory to rely on OPTSAT, otherwise all the modifications explained in the Section above are ignored, and SATPLAN$^{\prec}$ behaves like the standard SATPLAN.
-opt: specifies if the optimization is related to the number of actions, or to "soft" goals ("action" or "goal").
-minimality: specifies the type of optimization, i.e., if the optimization is subject to cardinality or subset inclusion ("card" or "subset").
-enc: if minimality is by cardinality, this option specifies what encoding has to be used ("w" or "b")
-problem: is the planning problem in (propositional) STRIPS format
-domain: is the domain specification in (propositional) STRIPS format

It should be noted that only the last two options are mandatory, i.e., if only the problem and domain parameters are specified the other options are set by default to their first choice ("optsat1.0", "action", "card", and "w" respectively).

Otherwise, a command line like the following:

./SATPLAN$^{\prec}$ -opt goal -minimality subset -problem log.pddl -domain domain-log.pddl

specifies that a logistic planning problem is subject to a maximization (under subset inclusion) of the "soft" goals to be satisfied. Given that no encoding is involved, the related specification (i.e., "-enc") is in this case useless. The OPTSAT system is used as a back-end solver for SATPLAN$^{\prec}$.

5 Experimental evaluation

We remind that SATPLAN can only handle propositional STRIPS domains, and, among them, we considered the pipesworld, satellite, airport, promela philosophers and optical, psr, depots, driverLog, zenoTravel, freeCell, logistic, blocks, mprime and mistery domains from the first 4 IPCs, and pathway, storage, tpp and trucks from IPC-5 (notice that we do not consider the domains in the "simple preferences" track in IPC-5 because they are not handled by SATPLAN). These are standard planning problems in which the goal corresponds to a set G of goals and without soft goals. We modified these problems in order to interpret all the goals in G as soft goals, following what is explained in Section 3 for SATPLAN$^{\prec}$ about goal selectors, and by encoding the problems in the language PDDL3 [13] for SGPLAN. Note that our proposed modification exactly encodes the fact that the goals in G are now "soft" in the sense that it is desirable but not necessary to achieve them. Since there are no "hard" goals, the various versions of SATPLAN/SATPLAN$^{\prec}$ would always find a valid plan, even when the makespan n is 0 (in which case the returned plan would be the empty one). In order to avoid this situation, we added a constraint stating that at time n at least one of the soft goals has to be satisfied. Because of this, we discarded the problems whose original version has only one goal because they would have no soft goal.[1] In the following, we use SATPLAN$^{\prec}$(s), SATPLAN$^{\prec}$(w) and SATPLAN$^{\prec}$(b) to denote SATPLAN$^{\prec}$ when is subject to subset inclusion optimality, and to cardinality (with W-encoding and B-encoding, respectively). We also use SATPLAN and SATPLAN(m) to denote basic SATPLAN employing SIEGE and MINISAT respectively. We considered both SAT solvers because (i) SIEGE is the default solver for SATPLAN, but (ii) MINISAT is used in SATPLAN$^{\prec}$ and OPTSAT because SIEGE's code is not available, and, most importantly, MINISAT is the winner of the last SAT Competition, in 2005,[2] together with the SAT/CNF minimizer SATELITE, and the winner of the SAT race 2006.[3] In any case, in our experiments, we have seen no significant differences in SATPLAN's performances when employing the two solvers.

In [11] we showed that SATPLAN$^{\prec}$ is competitive with respect to SGPLAN in terms of both metric (in this case the number of soft goals satisfied), and number of problem solved, i.e., with respect to the first two parameters used to evaluate planning systems in the "SimplePreferences"-track of the last IPC-5. Moreover, in the same paper, it is shown that this is achieved without sacrificing efficiency, i.e., that only in a small

[1] Note that also SGPLAN does not accept problems with only one goal.
[2] `http://www.satcompetition.org/2005/`
[3] `http://fmv.jku.at/sat-race-2006/`

fraction of the hundreds of problems analyzed SATPLAN$^{\prec}$ (with the W-encoding in case of optimality by cardinality) can have performances significantly worse than SATPLAN when minimizing the number of actions in the plan.

On the contrary, in this paper we focus on:

- the reduction in the number of actions in the plan returned by SATPLAN$^{\prec}$ over SATPLAN (and SGPLAN);
- the number of soft goals satisfied in the plan of SATPLAN$^{\prec}$ over SATPLAN;
- a comparison between SATPLAN$^{\prec}$, SATPLAN and SGPLAN on the makespan returned, on problem instances in which all the systems satisfy all soft goals.

Even if the real contribution of the analysis we will be showing is over SATPLAN, we decided also to include SGPLAN as a reference, taking into account that it uses a different approach, and that it is not targeted for optimizing the number of actions and makespan in the returned plan.

All the tests have been run on a Linux box equipped with a Pentium IV 2.4GHz processor and 512MB of RAM. The timeout has been set to 300s for each problem instance. In the plots, a point is missed if a system runs out of time or memory in the given problem instance.

The first results are shown in Figure 1. The left plot shows the performances of SATPLAN, SATPLAN(m), SATPLAN$^{\prec}$(w) and SATPLAN$^{\prec}$(s) on the problems considered, ordered according to SATPLAN's performances. The plot shows the number of soft goals each planner does not satisfy. This way of presenting the data has the feature that results about the same problem are not lost, i.e., the results for all the systems at a given x-coordinate refer to the same problem.

As expected SATPLAN does not satisfy many of the soft goals, in particular when using MINISAT, while SATPLAN$^{\prec}$(w)/(s) manage to satisfy all of them in many cases. Interestingly, the number of soft goals not satisfied by SATPLAN(w)/(s) are in most cases equal, while in theory this is not necessary the case. In the plot, for sake of readability, we have not included SATPLAN$^{\prec}$(b): its plot would be exactly the same to the one of SATPLAN$^{\prec}$(w), given that they manage to solve all the benchmarks presented, and with similar performances. Moreover, the performances of SATPLAN$^{\prec}$(w)/(b) are very similar to the ones of SATPLAN$^{\prec}$(s), SATPLAN and SATPLAN(m). The explanation for the CPU performances of SATPLAN$^{\prec}$(w)/(b) is that all the problems considered have at most 30 soft goals and the burden introduced by the Boolean encodings is negligible. We will see that, when considering minimal #actions-plan, this is no longer the case. Another interesting observation can be drawn from the use of the SAT solvers in SATPLAN: MINISAT manages to always satisfy very few soft goals (almost always one). Given this, and given that SATPLAN$^{\prec}$ is based on MINISAT, the results on SATPLAN$^{\prec}$ are made even stronger by this point.

Considering the "quality" of the plan returned in terms of number of actions, the results are in Figure 1, right plot. The results are again ordered according to SATPLAN's performances.

In the plot, for sake of readability, we do not show the results for the airport, promela philosophers and optical, and psr domains: for each problem in these domains SATPLAN, SATPLAN(m) and SATPLAN$^{\prec}$/(w)/(b)/(s) return plans with the same length. Sim-

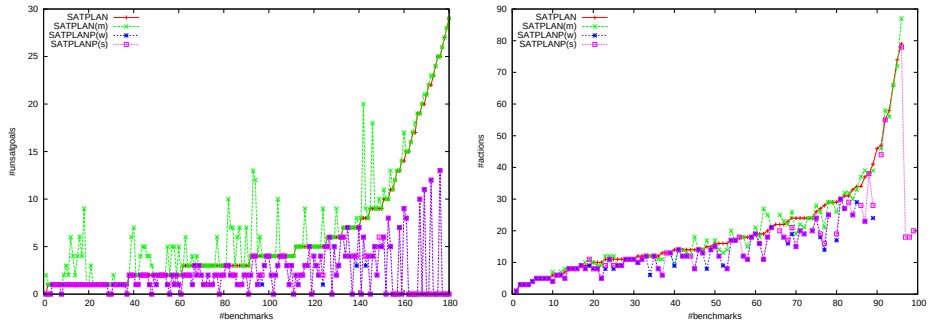

Fig. 1. Left: Number of unsatisfied soft goals by SATPLAN, SATPLAN(m), and SATPLAN$^{\prec}$(w)/(s). Right: Number of actions in the returned plan for SATPLAN, SATPLAN(m), and SATPLAN$^{\prec}$(w)/(s).

ilarly to the case of soft goals, the quality of the plan returned by SATPLAN(s) is in most cases equal to that of SATPLAN(w), and in many cases both return plans of better quality than SATPLAN. Again, in the plot we do not include results for SATPLAN$^{\prec}$(b): when it manages to solve a problem, it obviously returns a plan of the same quality of SATPLAN$^{\prec}$(w), but in several cases it runs out of time or memory when problems are big. Thus, its representative line would be exactly the same of SATPLAN$^{\prec}$(w), but with a significant portion of the line missing.

We remind that in [11] we showed that SATPLAN$^{\prec}$ is competitive with respect to SGPLAN in terms of both number of soft goals satisfied and number of problem solved, i.e., with respect to the first two parameters used to evaluate planning systems in the "SimplePreferences"-track of the last IPC-5, without sacrificing efficiency.

Now, we want to evaluate what is the behavior, and the relative performance, of SGPLAN with respect to SATPLAN$^{\prec}$ on the metrics for which SATPLAN$^{\prec}$ is optimized, i.e., number of actions and makespan of the returned plan.

This analysis is performed on those instances in which all the systems satisfy all the soft goals (when greater than one),[4] i.e., we posed SGPLAN and all the versions of SATPLAN/SATPLAN$^{\prec}$ on the same conditions. Table 1 shows the results obtained and is structured as follows: the first column contains the specific instance (where sat, air, driv, zeno, free, log, block, and stor stay for satellite, airport, driverLog, zeno-Travel, freeCell, logistics, blocks-world and storage domain, respectively); columns 2-6 contain the number of actions in the returned plan for SATPLAN, SATPLAN(m), SATPLAN$^{\prec}$(w), SATPLAN$^{\prec}$(s) and SGPLAN, respectively, while the last two columns contain the makespan for all the modification of SATPLAN/SATPLAN$^{\prec}$ and SGPLAN,

[4] We also could not consider some domains, e.g., promela philosophers, because SGPLAN time outs or returns segmentation faults on all instances, the latter due to the high number of grounded operators in the problems. The last information is a personal communication with the authors.

	#actions					Makespan	
PB	SATPLAN	SATPLAN(m)	SATPLAN$^{\prec}$(w)	SATPLAN$^{\prec}$(s)	SGPLAN	SATPLAN[$^{\prec}$]	SGPLAN
log4-0	24	24	20	20	20	9	20
log4-1	24	26	19	21	19	9	19
log4-2	24	19	15	15	15	9	15
log5-0	31	32	27	27	31	9	31
log5-1	29	26	17	19	17	9	17
log6-0	33	30	25	25	26	9	26
log6-1	28	21	14	16	15	9	15
log6-9	41	39	24	28	28	11	28
block4-2	6	6	6	6	6	6	6
block5-2	16	16	16	16	26	16	26
block6-0	12	12	12	12	12	12	12
block6-1	10	10	10	10	16	10	16
block6-2	20	20	–	20	32	20	32
stor5	11	9	9	9	8	6	8
stor6	11	9	9	9	10	6	10
stor7	14	14	14	14	14	14	14
stor8	16	14	12	12	13	8	13
stor9	14	14	12	12	11	7	11
stor10	–	–	–	18	18	18	18
stor11	–	–	–	18	17	11	17
stor12	22	25	–	20	17	9	17
sat1	9	9	9	9	9	8	9
sat2	13	–	–	13	13	12	13
psr15	10	10	10	10	10	8	10
psr19	25	25	25	25	31	15	31
psr25	9	9	–	9	9	9	9
psr33	21	21	21	21	21	16	21
psr40	20	20	20	–	20	15	20
psr42	30	30	30	30	30	16	30
driv1	14	18	8	8	7	6	7
zeno2	6	7	6	6	8	5	8
zeno3	13	11	6	6	6	5	6
zeno4	11	11	11	11	13	5	13
zeno5	15	14	14	14	11	5	11
zeno6	14	13	12	12	13	5	13
zeno8	16	17	15	15	12	5	12
zeno9	29	29	–	–	23	6	23
free1	9	11	9	11	10	5	10
free2	18	18	–	18	14	8	14
free3	21	21	21	21	19	7	19
tpp4	14	14	14	14	14	5	14
tpp5	19	21	19	19	19	7	19
air7	41	41	41	41	41	21	41
air9	71	71	–	71	73	27	73
air12	39	39	39	39	39	21	39

Table 1. Number of actions and makespan for SATPLAN, SATPLAN$^{\prec}$ and SGPLAN.

respectively. An "$-$" indicates that the corresponding instance has not been solved in the given time limit of memory limit. The results suggest the following considerations:

- The makespan returned by SGPLAN if often significantly higher than the one returned by the various versions of SATPLAN and SATPLAN$^{\prec}$, up to a factor of 4 (zeno9 instance). Nonetheless, there are also (even if just a few) cases in which SGPLAN returns the same, optimal makespan (namely psr25, block4-2, block6-0, stor4 and stor10).
- When a reduction on the number of action is possible (i.e., given the fixed makespan and the "structure" of the planning instance) SATPLAN$^{\prec}$ can return plans with a significant lower number of actions, with respect to both SATPLAN and SGPLAN: many of the logistics and blocks-world instances better underline this behavior. Interestingly, there are also instances where SGPLAN returns plans with fewer actions than SATPLAN/SATPLAN$^{\prec}$, e.g., driv1, zeno5, free2, and stor11). This is due to the usual non-optimal makespan returned by SGPLAN, and from the intuitive consideration that on some instances, with a longer makespan actions can be easily better "serialazable", thus producing a plan with fewer actions. In fact, finally note how in the instances where this happen, SGPLAN returns a much higher makespan than SATPLAN/SATPLAN$^{\prec}$ (but driv1).

The last point opens the way to an interesting research issue, suggesting a possible extension of SATPLAN$^{\prec}$ in order to further improve its plan quality. The idea would be to "trade-off" between optimality of the makespan and number of actions in the plan. We have seen that, on some instances, it is possible to compute plans of better quality by do not stop the search when the (first) plan is found, but going ahead and increase the makespan. It is interesting to note how the same trade-off could also help when trying to maximize the number of soft goals satisfied.

6 Conclusions

In this paper we have presented a system, SATPLAN$^{\prec}$, which makes a significant step toward the elimination of same deficiency of SATPLAN related to plan quality, i.e., the number of actions and the satisfied soft goals in the returned plan. We have considered a wide number of benchmarks from all previous IPCs, and experimentally shown that significant gain can be obtained with SATPLAN$^{\prec}$. Interestingly, this is often achieved without sacrificing efficiency, and being competitive with SGPLAN, the winner of the "SimplePreferences"-track of the last IPC-5. As a future work, we plan to analyze if the results reported in this paper can be further strengthened by not considering a bounded horizon, and what is the corresponding lost in the efficiency of the system.

7 Availability of the system and SGPLAN benchmarks

The binary executable of SATPLAN$^{\prec}$, together with the benchmarks used adapted to the PDDL3 format in order to be processed by SGPLAN, are available at
`http://www.star.dist.unige.it/~marco/SATPLANP/`.

References

1. Kautz, Henry and Selman, Bart. Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.
2. Kautz, Henry and Selman, Bart. Unifying SAT-based and graph-based planning. In *Proc. IJCAI-99*, pages 318–325, Morgan-Kaufmann, 1999.
3. Henry Kautz and Bart Selman. SATPLAN04: Planning as satisfiability, In *Proc. of 5th International Planning Competition, Internation Conference on Automated Planning and Scheduling (ICAPS-06)*, pages 45–47, 2006.
4. Hoffmann, Joerg and Edelkamp, Stefan. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research (JAIR)*, 24:519–579, 2005.
5. Z. Xing, Y. Chen and W. Zhang. MaxPlan: Optimal planning by decomposed satisfiability and backward reduction. In *Proc. of 5th International Planning Competition, Internation Conference on Automated Planning and Scheduling (ICAPS-06)*, pages 53–55, 2006.
6. E. Giunchiglia and M. Maratea. OPTSAT: A Tool for Solving SAT related optimization problems. In *Proc. of the 10th European Conference on Logics in Artificial Intelligence (JELIA)*, pages 485–489, LNCS 4160, Springer, 2006.
7. E. Giunchiglia and M. Maratea. Solving Optimization Problems with DLL. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI)*, pages 377–381, IOS Press, 2006.
8. C. Hsu and B. Wah and R. Huang and Y. Chen. Constraint Partitioning for Solving Planning Problems with Trajectory Constraints and Goal Preferences. In *Proc. IJCAI*, pages 1924–1929, 2007.
9. J. P. Warners. A linear-time transformation of linear inequalities into CNF. *Information Processing Letters*, 68(2):63–69, 1998.
10. Olivier Bailleux and Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In *Proc. 9th International Conference on Principles and Practice of Constraint Programming (CP-03)*, LNCS, Springer, 2003.
11. E. Giunchiglia and M. Maratea. Planning as Satisfiability with Preferences. *Proc. of 22th National Conference of the American Association for Artificial Intelligence (AAAI)*, to appear, 2007.
12. M. Büttner and J. Rintanen. Satisfiability planning with constraints on the number of actions. In *Proc. ICAPS*, pages 292–299, 2005.
13. A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. In *Proc. of 5th International Planning Competition, Internation Conference on Automated Planning and Scheduling (ICAPS-06)*, pages 7–13, 2006.