Exploiting Macro-actions and Predicting Plan Length in Planning as Satisfiability

Alfonso Gerevini, Alessandro Saetti, and Mauro Vallati

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia
Via Branze 38, 25123 Brescia, Italy
{gerevini, saetti, mauro.vallati}@ing.unibs.it

Abstract. The use of automatically learned knowledge for a planning domain can significantly improve the performance of a generic planner when solving a problem in this domain. In this work, we focus on the well-known SAT-based approach to planning and investigate two types of learned knowledge that have not been studied in this planning framework before: macro-actions and planning horizon. Macro-actions are sequences of actions that typically occur in the solution plans, while a planning horizon of a problem is the length of a (possibly optimal) plan solving it. We propose a method that uses a machine learning tool for building a predictive model of the optimal planning horizon, and variants of the well-known planner SatPlan and solver MiniSat that can exploit macro actions and learned planning horizons to improve their performance. An experimental analysis illustrates the effectiveness of the proposed techniques.

Keywords: Machine learning for planning. Planning as satisfiability.

1 Introduction

Learning for planning is an important research field in automated planning research, that, as demonstrated by the last two planning competitions [6, 3], in the recent years has received considerable attention in the planning community. Starting from the PDDL formalization of a planning problem, the current learning techniques for deterministic (classical) planning aim at automatically generating additional knowledge about the problem, and at effectively using it to improve the performance of a planner.

In this paper, we consider two types of learned knowledge for optimal planners in the "planning as satisfiability" framework (also called SAT-based planning) [11]: *macro-actions* and the planning *horizon*. Macro-actions are (usually short) sequences of actions that typically occur in the plans solving the problems of a given planning domain. The planning horizon of a planning problem is the length of a (possibly optimal) plan solving the problem, that for classical planning is defined as the number of time steps in the plan.

Regarding macro-actions, several systems for learning and using them in a planner have been developed, e.g., [2, 16]. However, to the best of our knowledge, the use of macro-actions in SAT-based planning has never been investigated. Regarding learned horizons, we are not aware of any existing work about learning this information and exploiting it in SAT-based planning.

We focus our study on SatPlan [11–13], one of the most popular and efficient SAT-based optimal planning system. Essentially, first SatPlan uses a preprocessing algorithm to compute a lower (possibly exact) bound k on the optimal planning horizon, and translates the planning problem into a SAT problem, i.e., the satisfiability of a propositional formula in CNF (shortly a CNF) encoding the problem. If the SAT problem is solvable (the CNF is satisfiable), a plan with at most k time steps can be derived from a model of the CNF. If the SAT problem is unsolvable (the CNF is unsatisfiable), SatPlan generates a larger SAT problem using an increased bound, and so on, until it finds a solution or it proves that the original planning problem has no solution.

While SatPlan can use any SAT solver, in this work we concentrate our study on MiniSat [5], a very well-known efficient solver based the DPLL algorithm [4], extended with backtracking by conflict analysis and clause recording [14] and with boolean constraint propagation (BCP) [15].

The paper contains the following contributions in the context of SatPlan: (i) a new variant of MiniSat that can exploit a given set of macro-actions to improve the performance of SAT solving for a CNF encoding a planning problem, (ii) a machine learning technique for constructing a predictive model of the optimal planning horizon for a given problem, and (iii) the use of this model, possibly in combination with macro-actions, to reduce the number of SAT problems during planning. The effectiveness of these techniques is studied in a preliminary experimental analysis, showing that they can lead to significant performance improvements.

The rest of the paper is organized as follows. Section 2 describes a modified version of MiniSat exploiting macro-actions; Section 3 introduces a method for estimating the optimal planning horizon, and proposes another enhanced version of MiniSat for computing shorter plans from a satisfiable CNF; Section 4 presents our experimental results; finally, Sections 5 gives the conclusions.

2 Using Macro-actions during SAT Solving

In order to exploit macro-actions in the SAT-solver of SatPlan, we have modified the well-known solver MiniSAT by using macro-actions to bias the way in which the propositional variables are processed (i.e., selected and instantiated). The intuitive general idea is giving preference to unassigned variables corresponding to actions that would include in the current plan macro actions that are compatible with the current assignment. We shall illustrate the idea with an example after introducing more precisely the notion of macro-action for SatPlan. In the rest of the paper, variable v_i^j of the CNF encoding a planning problem denotes action a_i planned at time step j.

In the context of SatPlan, a macro-action is a sequence of propositional variables representing actions executed at certain time steps. For example, consider a planning problem in the well-known `BlocksWorld` domain, and assume that $\langle a_1, a_2 \rangle$ is a macro-action for this problem, where a_1 and a_2 abbreviate actions `(pick-up A)` and `(stack A B)`, respectively. Moreover, suppose that the planning horizon (plan steps) in the SAT problem encoding the planning problem under considerations is 5, and the earliest time steps where a_1 and a_2 can be planned are 3 and 4, respectively. Then the CNF encoding the planning problem contains 3 variables v_1^3, v_1^4 and v_1^5 representing action `(pick-up A)` planned at time steps 3, 4 and 5, respectively, and 2 variables v_2^4 and

MacroMiniSAT($\mathcal{F}$, M)

Input: The CNF $\mathcal{F}$ encoding a planning problem, a set M of macro actions.

Output: A solution variable assignment W or *failure*.

1. $W \leftarrow \emptyset$;
2. **while** $\exists$ variable v of $\mathcal{F}$ not assigned in W **do**
3. $v \leftarrow$ SelectVariableFromMacros($\mathcal{F}$, W, M);
4. **if** $v = nil$ **then** $v \leftarrow$ SelectVariable($\mathcal{F}$, W);
5. $W \leftarrow W \cup (v = \mathsf{TRUE})$;
6. *Propagate value of* v;
7. **if** the propagation has generated conflicts **then**
8. **if** the propagation has generated a top level conflict **then return** *failure*;
9. **else** *Perform backtracking*;
10. **return** W.

Fig. 1: MiniSAT modified for solving the SAT encoding of a planning problem using macro-actions.

v_2^5 representing action `(stack A B)` planned at time steps 4 and 5, respectively. For this CNF, two possible macro-actions are $\{v_1^3, v_2^4\}$ and $\{v_1^4, v_2^5\}$. If, for instance, in the current assignment of the SAT-solver v_2^5 is true, v_2^4 is false, and the other variables are unassigned, then v_1^4 is preferred to v_1^3.

Figure 1 gives a high-level description of a variant of MiniSAT, called MacroMiniSAT, for solving the SAT encoding $\mathcal{F}$ of a given planning problem using a set M of macro-actions. Initially, the current set of assigned variables (W) is empty (step 1). At each iteration of the loop 2–9, an unassigned variable v of the input CNF $\mathcal{F}$ is selected by either procedure SelectVariableFromMacros or procedure SelectVariable, and the value of the selected variable is set to true (steps 3–5). Each variable selection and instantiation is called a (search) *decision*. Then, the effects of the last decision are propagated by unit propagation (steps 6–9): when a clause becomes unary under the current assignment, the remaining literal in the clause is set to true and this decision is propagated, possibly reducing other clauses to unary clauses and repeating the propagation. The propagation process continues until no more information can be propagated. If a conflict is encountered (all literals of a clause are false), a conflict clause is constructed and added to the SAT problem. The decisions made are canceled by backtracking, until the conflict clause becomes unary. This unary clause is propagated and the search process continues.

The main difference w.r.t. MiniSAT concerns SelectVariableFromMacros($\mathcal{F}, W, M$), which uses a macro m selected from a set $M_A \subseteq M$ of macros to determine the next variable to instantiate. A macro m' is in M_A if the three following conditions hold:

1. At least one variable of m' is unassigned;
2. All the assigned variable of m' are true according to the current variable assignment W;
3. m' contains no variable belonging to another macro m'' formed by only variables that are true according to W.

The rationale of condition 2 is to avoid preferring variables representing actions that, given the current variable assignment, will not appear in the solution plan. The motivation of condition 3 is based on the empirical observation that, for many domains, often two macro-actions sharing one or more actions do not appear simultaneously in a solution plan.

If set M_A contains more than one macro, SelectVariableFromMacros prefers the macro m' in M_A according to the actions in a given *relaxed plan* π (see, e.g., [9, 10, 7, 17]) for the planning problem under consideration. Plan π is relaxed in the sense that it does not consider the possible negative interference between planned actions, and it can be quickly computed by a polynomial algorithm (see, e.g., [10]). Essentially, SelectVariableFromMacros chooses a variable from the macro m' in M_A formed by the highest percentage of variables of m' representing actions in π.

For example, consider a `BlocksWorld` problem in which `A` and `B` are on the table, `C` is on `B`, and the goal is moving `A` on `B`. The following sequence of actions is a possible relaxed plan for our running example: `(unstack C B)`, `(pick-up A)`, `(stack A B)`. In the original planning problem, action `(unstack C B)` would make the robot arm occupied, but, since this is represented through a negative effect of the action, in the relaxed plan the arm remains free after the execution of `(unstack C B)`, and so action `(pick up A)` can be planned.

Assume that sequences $\langle a_1, a_2 \rangle$ and $\langle a_3, a_4 \rangle$ are two macro-actions for this `BlocksWorld` problem, where a_1, a_2, a_3 and a_4 abbreviate `(pick-up A)`, `(stack A B)`, `(pick-up C)`, `(stack C B)`. Moreover, assume that v_2^5 is false in the current variable assignment. Then, M_A is formed by $\{v_1^3, v_2^4\}$, $\{v_3^3, v_4^4\}$ and $\{v_3^4, v_4^5\}$; the percentage of variables in these macros representing actions in the relaxed plan is 100, 0 and 0, respectively. Therefore, SelectVariableFromMacros chooses macro $\{v_1^3, v_2^4\}$.

If the number of macro-actions with the highest percentage of variables representing actions in the relaxed plan is greater than one, then SelectVariableFromMacros uses some secondary criteria to select the most promising macro. These criteria include the ratio between the number of variables assigned as true and the cardinality of the macro, the sum of the variable *activity values* (as defined in [5]), and the time step of the first action in the macro. If none of the these criteria returns a single macro, SelectVariableFromMacros randomly chooses a macro from the set of the best macros. Finally, it returns the earliest (time-step wise) unassigned variable from the best macro.

If set M_A contains no macro, SelectVariableFromMacros returns *nil* and, subsequently, the algorithm uses the standard MiniSAT procedure SelectVariable (as defined in [5]) for choosing an unassigned variable of the CNF.

3 Predicting and Using Learned Horizons in SATPLAN

Typical SAT-based planners like SatPlan generate several unsatisfiable CNF encodings of the given planning problem with different (increasing) plan length bounds before finding a solvable CNF (from which an optimal plan is obtained). Unfortunately, we have observed that, while for a solvable CNF the use of macro-actions can speed up the SAT-solver, often for an unsolvable CNF this is not the case. Hence, in order to better exploit the macro-actions in the whole planning process, we have developed a method for predicting the optimal planning horizon of a problem in a given domain. It should be

Name	Description
$\#G$	number of problem goals
$\#O$	number of problem objects
$\#F$	number of facts in the initial state
$\#A$	number of actions grounded by planner LPG
$\#LM$	number of landmarks computed by planner Lama
$\#ME$	number of mutex exclusive relations computed by LPG
π_r^{FD}	number of actions in the relaxed plan constructed by planner FastDownward
π_r^{FF}	number of actions in the relaxed plan constructed by planner FF
π_r^{Lama}	number of actions in the relaxed plan constructed by Lama
π_r^{LPG}	number of actions in the relaxed plan constructed by LPG

Table 1: The set of features used to define a predictive model of the planning horizon.

noted that such a predictive model is an independent technique that can be used without the macro-actions.

The predictive model is constructed using a set of features, given in Table 1, concerning the planning problem, the mutex relations and landmarks in its state space (e.g., [7, 17]), and the relaxed solutions used in some heuristics of state-of-the-art satisficing planners. The values of features $\#G$, $\#O$, $\#F$ are derived from the "grounded" description of the planning problem, while the values of the other features are derived using the planning techniques implemented in FastDownward [9], FF [10], Lama [17] and LPG [7]. Essentially, for each training problem Π, the length (number of plan steps) l_π of an optimal solution π for Π is computed using an existing optimal planner; the values of the learning features for Π and l_π provide the data from which the predictive model is generated using a machine-learning tool. In our implementation, we used the well-known tool WEKA [18] with technique M5Rules [8].

The experimental results in Table 2 indicate that, for problems with short optimal plans, the estimated optimal horizon computed by WEKA is sometimes better than the first horizon computed by SatPlan, which is the initial length of Graphplan's planning graph; while for problems with middle-size and long optimal plans, the estimated optimal horizons are always better. Moreover, these results indicate that the predicted plan length can be higher than the actual optimal plan length, while SatPlan's initial horizon is a lower or exact bound. Therefore, in order to use a learned horizon in SatPlan, we have modified its standard behaviour as follows.

If the initial CNF $\mathcal{F}$ encoding the planning problem with a predicted horizon t is solvable, then the process is repeated using a CNF with horizon $r-1$, where r is the number of time steps in the solution plan computed from $\mathcal{F}$, and so on, until a horizon $q < t$ for which the CNF is unsolvable has been identified (the optimal solution plan is the one generated from the CNF with horizon $q+1$). Otherwise, the process is repeated with horizon $t+1$, and the planner stops when a solvable CNF is generated (the optimal solution is the plan derived from the solution of this last CNF).

Each generated CNF can be solved using the modified version of the MacroMiniSAT procedure shown in Figure 2. The gray steps indicate the differences with respect to the version using only macro actions given in Figure 1. At each iteration of the loop 2–13, if the selected variable v represents an action at level (time step) l, procedure

Problem	Opt. Length	Δ_G	Δ_W
Short plans			
BlocksWorld – 6	8	-2	2
Depots – 3	9	-4	1
Ferry – 3	7	-3	3
Goldminer – 5	8	0	5
Gripper – 4	7	-4	1
Matching-BW – 5	7	-3	4
Sokoban – 6	8	0	2
Middle-size plans			
BlocksWorld – 8	18	-8	-1
Depots – 5	12	-3	2
Ferry – 4	13	-9	1
Goldminer – 7	16	-3	0
Gripper – 8	15	-12	1
Matching-BW – 8	15	-8	0
Sokoban – 7	19	-6	-5
Long plans			
BlocksWorld – 28	81	-47	3
Depots – 10	20	-6	4
Ferry – 11	40	-36	0
Goldminer – 15	92	-18	16
Gripper – 11	23	-20	-1
Matching-BW – 30	46	-28	2
Sokoban – 12	51	-31	4

Table 2: Empirical evaluation of the accuracy of the predicted planning horizon using some problems with different size from 7 known domains: optimal plan length (2nd column), gap between the first plan length bound of SatPlan (defined as the length of the initial Graphplan's planning graph) and the optimal plan length (3rd column), and gap between the predicted planning horizon and the optimal plan length. Smaller Δ-values indicate better estimates.

PropagateGoalNoop is called to possibly assign true to the unassigned variables encoding "no-ops" at every time step $i > l$ and representing problem goals. As observed below, this is useful when the solution plan has a length that is lower that the current plan horizon.

The loop 4–11 of PropagateGoalNoop assigns true to each unassigned variable v representing a problem goal at level l, and propagates this decision. If a conflict is generated, and backtracking is performed. The outer loop (steps 2–11) repeats these variable assignments for each time step from the input time step l to the latest time step of an action encoded in the input CNF $\mathcal{F}$.

We experimentally observed that this modified version of MacroMiniSAT generates plans shorter than those generated by the version in Figure 1. Hence, when the pre-

MacroMiniSAT($\mathcal{F}, M$)

Input: The CNF $\mathcal{F}$ encoding a planning problem, a set M of macro actions.

Output: A solution variable assignment W or *failure*.

1. $W \leftarrow \emptyset$;
2. **while** $\exists$ variable v of $\mathcal{F}$ not assigned in W **do**
3. $v \leftarrow$ SelectVariableFromMacros($\mathcal{F}$, W, M);
4. **if** $v = nil$ **then** $v \leftarrow$ SelectVariable($\mathcal{F}$, W);
5. $W \leftarrow W \cup (v = \mathsf{TRUE})$;
6. *Propagate value of* v;
7. **if** the propagation has generated no conflict **then**
8. **if** v represents an action **then**
9. $W \leftarrow$ PropagateGoalNoop($\mathcal{F}$, W, $Level(v) + 1$);
10. **if** $W =$ *failure* **then return** *failure*;
11. **else**
12. **if** the propagation has generated a top level conflict **then return** *failure*;
13. **else** *Perform backtracking*;
14. **return** W.

PropagateGoalNoop($\mathcal{F}, W, m$)

Input: The CNF $\mathcal{F}$ encoding a planning problem, a variable assignment W, and a time step m.

Output: A (partial) variable assignment W or *failure*.

1. **if** $m >$ *EndLevel* **then return** W;
2. **for** $l = m$ **to** *EndLevel* **do**
3. $G(l) \leftarrow$ set of variables of $\mathcal{F}$ encoding no-ops at level l and representing goals;
4. **foreach** unassigned variable v in $G(l)$ **do**
5. $W \leftarrow W \cup (v = \mathsf{TRUE})$;
6. *Propagate value of* v;
7. **if** the propagation has generated no conflict **then**
8. **if** all variables of $\mathcal{F}$ are assigned **then return** W;
9. **else**
10. **if** the propagation has generated a top level conflict **then return** *failure*;
11. **else** *Perform backtracking*;

Fig. 2: Algorithms for solving the SAT encoding of a planning problem using a set of macro-actions when the horizon can be higher than the optimal one. Function *Level*(v) returns the time step of the action encoded by variable v. *EndLevel* represents the latest time step of an action encoded in the input CNF $\mathcal{F}$.

dicted horizon is greater than the optimal one, SatPlan generates (and solves through MacroMiniSAT) fewer CNFs.

4 Experimental Results

In this section, we give some results from an experimental analysis aimed at

– understanding the effectiveness of using macro-actions for SAT-based planning;

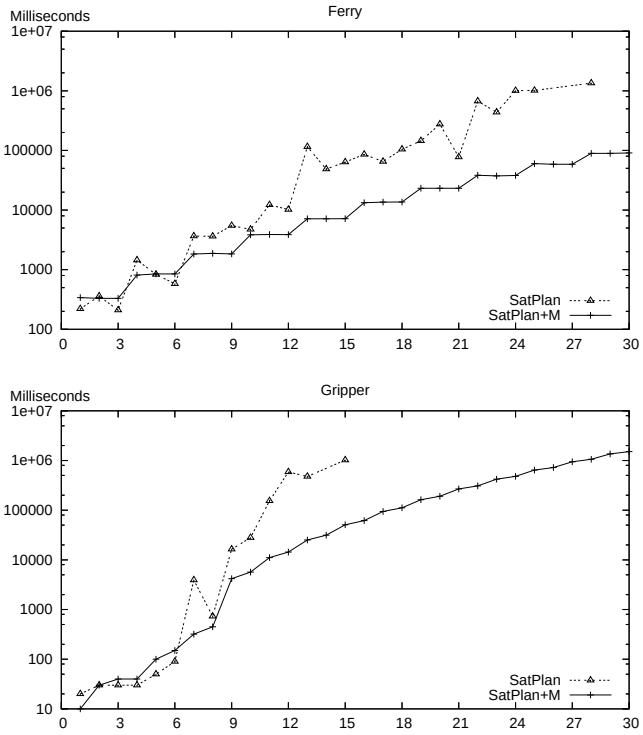

Fig. 3: CPU time of SatPlan and SatPlan+M to solve the SAT problems encoded from the planning problems of domains `Ferry` and `Gripper` using the optimal horizon. On the x-axis we have the problem names simplified by numbers.

– evaluating the impact of using learned horizons instead of those computed by SatPlan.

All experimental tests were conducted using an Intel Xeon(tm) 3 GHz machine, with 2 Gbytes of RAM. Unless otherwise specified, the CPU-time limit for each run was 30 minutes, after which termination was forced. (The CPU time used for computing the values of the considered problem features and estimating the plan length is generally negligible w.r.t. the time used for planning, and in our analysis it is ignored.) The CPU time used for computing the values of the problem features and estimating the plan length is generally negligible w.r.t. the time used for planning, and is ignored in our analysis.) In our experiments, macro-actions were computed using the techniques incorporated into Macro-FF [2], a well-known available planning system. However, any other system for computing macro-actions could be used.

Overall, the experiments consider 7 known planning domains: `BlocksWorld`, `Depots`, `Ferry`, `Goldminer`, `Gripper`, `Matching-BW` and `Sokoban`. However, for testing the use of macro-actions we focus only on domains `Ferry` and `Gripper`. Domain `Ferry` concerns transporting cars between locations using a ferry. Each location is directly connected to every other location; cars can be debarked and boarded; the available ferry can carry at most one car at a time. The only macro-action used in our experiments

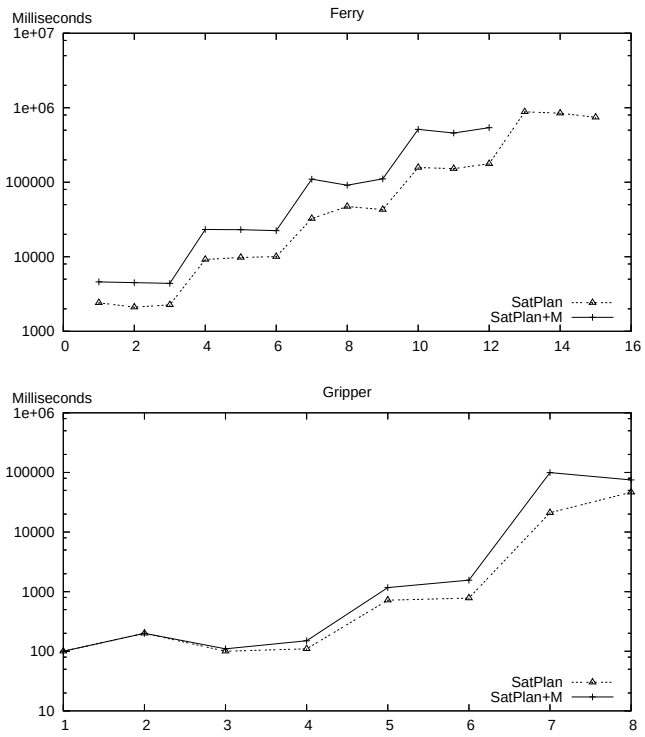

Fig. 4: CPU times of SatPlan and SatPlan+M in domains `Ferry` and `Gripper`. On the x-axis we have the problem names simplified by numbers.

for this domain is formed by three domain actions (no other macro-action is learned): boarding a car x from a location l_1 to the ferry, moving the ferry from l_1 to another location l_2, and debarking x in l_2. Domain `Gripper` concerns transporting some balls between two rooms using a robot with two gripper hands. The macro-action used in our experiments for this domain is formed by five actions (again this is the only learned macro-action): picking a ball x from a room r_1 by the right hand, picking a ball y from room r_1 by the left hand, moving the robot from r_1 to another room r_2, dropping x in r_2, and dropping y to r_2.

For the other domains considered in our experiments, the macro-actions computed by Macro-FF are not suitable for SatPlan because they are long and can appear only in sub-optimal plans, or there is a large number of similar macro-actions, involving many variables of the CNF. In these situations, the variant of MiniSat that tries to use macro-actions is not more efficient than the original version.

Figure 3 shows the CPU times required by the original version of SatPlan and the version using macros (abbreviated with SatPlan+M) for domains `Ferry` and `Gripper` when the optimal planning horizon is given (and the first generated SAT problem is solvable).

Domain	#Prob	% Solved		Mean CPU Time		Speed score	
		SatPlan	SatPlan+H	SatPlan	SatPlan+H	SatPlan	SatPlan+H
BlocksWorld	60	80	98.33	330.2	321.5	20.6	58.2
Depots	43	100	100	78.6	70.6	30.8	40.7
Ferry	18	83.3	94.4	223.5	176.5	4.9	17.0
Goldminer	44	97.3	95.5	464.5	374.9	31.0	41.3
Gripper	10	80.0	80.0	15.8	10.4	5.8	7.8
Matching-BW	30	96.7	96.67	228.5	146.9	16.9	28.5
Sokoban	140	99.0	100	232.3	93.2	61.0	138.5
Total	345	94.5	98.3	251.1	178.9	171.0	332.1

Table 3: Percentage of solved problems (columns 3-4), average CPU time (columns 5-6) and IPC score (columns 7-8) of SatPlan and SatPlan using the learned horizon estimated by WEKA for 7 known domains.

SatPlan+M is almost always faster than SatPlan and solves much larger problems. However, as shown in Figure 4, when the optimal planning horizon is not given, SatPlan+M is usually slower than SatPlan (up to about two times). The main reason why macro-actions slow down the planning process is that they are not useful when a SAT problem is unsolvable, and usually SatPlan generates several unsolvable SAT problems before the solvable one.

In order to better exploit macro-actions, it is important to have accurate bounds on the optimal horizon and minimize the number of the generated unsolvable SAT problems. In the last part of this section, we will show that using macro-actions can be useful when combined with the use of learned horizons (which often are upper bounds, rather than lower bounds as in SatPlan, on the optimal horizons).

Table 3 gives the percentage of solved problems, the average CPU time and the *speed score* of the original version of SatPlan and the proposed version using learned planning horizons (abbreviated with SatPlan+H). The speed score was first introduced and used by the organizers of the 6th International Planning Competition [6] for evaluating the relative performance of the competing planners, and since then it has become a standard method for comparing planning systems. The speed score of a system s is defined as the sum of the speed scores assigned to s over all the considered problems. The speed score assigned to s for a planning problem P is 0 if P is unsolved and $T^*_P/T(s)_P$ otherwise, where T^*_P is the lowest measured CPU time to solve problem P and $T(s)_P$ denotes the CPU time required by s to solve problem P. Higher values of the speed score indicate better performance.

The results in Table 3 indicate that the number of problems solved by SatPlan+H is greater than or equal to the number of those solved by SatPlan, and that SatPlan+H is almost always faster than SatPlan, because the speed score of SatPlan+H is very close to the number of considered problems.

Figure 5 shows the CPU time of the original version of SatPlan (which generates the first SAT problem using a lower bound on the optimal horizon through Graphplan) and our version using macros *and* the horizon learned by the proposed method (abbreviated with SatPlan+MH). The results in Figure 5 show that, for the considered

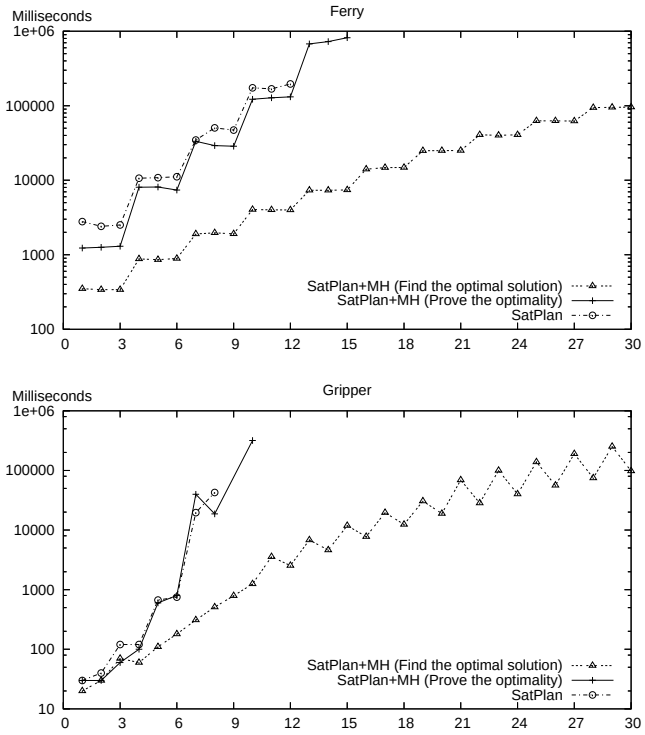

Fig. 5: CPU time of SatPlan and SatPlan+MH to find the optimal solution and to prove that such a solution is optimal for domains `Ferry` and `Gripper`. On the x-axis we have the problem names simplified by numbers.

domains, the combination of using learned horizons and macro-actions speeds up the planning process of SatPlan.

Finally, the CPU time of SatPlan+MH for finding the optimal plan is up to about two orders of magnitude lower than the time of SatPlan, and, moreover, SatPlan+MH solves many more problems. SatPlan+MH is often still faster than SatPlan in proving that a computed solution is optimal, but the performance gap is reduced. The reason why the performance gap is smaller is that, in order to prove optimality, one *un*solvable SAT problem (with horizon equal to the optimal horizon minus one) is generated and, as previously observed, for domains `Ferry` and `Gripper` the use of macro-actions for unsolvable SAT problems do not usually increase the performance.

5 Conclusions

We have investigated the use of macro-actions in SAT-based planning based on a variant of a well-known SAT solver that can exploit this information to speed up planning. Moreover, we have presented a new method for predicting the optimal planning horizon through a model generated using standard machine-learning techniques, and shown how to use it in combination with macro actions.

A preliminary experimental study indicates that: (i) the use of macro actions can speed up the SAT solver when the CNF encoding of the problem is satisfiable; (ii) the estimate of the optimal plan length computed by our method is more accurate than the bound computed by SatPlan through Graphplan's planning graph and it can be used to improve SatPlan speed; and (iii) the use of macro-actions combined with the learned planning horizon can speed up SatPlan.

Future work includes studying the use of macro-actions generated by other tools, e.g., WIZARD [16], and running additional experiments about the impact of macro-actions and learned horizons on the performance of SAT-based planning.

References

1. Blum, A., and Furst, M., L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.
2. Botea, A., Müller, M., and Schaeffer, J. 2005. Learning partial-order macros from solutions. In *Proc. of ICAPS-05*.
3. Celorrio, S. J., Coles, A. and Coles, A. 2011. 7th Int. Planning Competition – Learning Track. *http://www.plg.inf.uc3m.es/ipc2011-learning*.
4. Davis, M., Logemann, G., and Loveland, D. 1962. A machine program for theorem-proving. In *Communications of the ACM* 5:394–397.
5. Een, N., and Sörensson, N. 2003. An extensible SAT-solver. In *Proc. of SAT-03*.
6. Fern, A., Khardon, R., and Tadepalli, P. 2008. 6th Int. Planning Competition – Learning Track. *http://eecs.oregonstate.edu/ipc-learn/*.
7. Gerevini, A., Saetti, A., and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research* 20:239–290.
8. Hall, M.; Holmes, G; Frank, E. 1999. Generating Rule Sets from Model Trees. In *Proc. of Australian Conf. on AI, 1999*
9. Helmert, M. 2006. The Fast downward planning system. In *Journal of Artificial Intelligence Research* 26:191–246.
10. Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. In *Journal of Artificial Intelligence Research* 14:253–302.
11. Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proc. of ECAI-92*.
12. Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proc. of IJCAI-99*.
13. Kautz, H., Selman, B., and Hoffmann, J. 2006. SatPlan: Planning as satisfiability. In *Abstract Booklet of the 5th Int. Planning Competition*.
14. Marques, S., J., P., and Karem S., A. 1996. GRASP a new search algorithm for satisfiability. in *Proc. of ICCD-96*.
15. Moskewicz, M., W., and Madigan, C., F., and Zhao, Y., and Zhang, L., and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proc. of DAC-01*.
16. Newton, M., Levine, J., Fox, M., and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proc. of ICAPS-07*.
17. Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177.
18. Witten, I., H., and Frank, E. 2005. *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann. San Fransisco, CA.