Combining macros and SAT planning

Mauro Vallati*

Dipartimento di Ingegneria dell'Informazione,
Università degli Studi di Brescia,
Via Branze 38, 25123 Brescia, Italy
mauro.vallati@ing.unibs.it
http://www.ing.unibs.it/~mauro.vallati

Abstract. *Planning based on propositional satisfiability is a powerful approach for computing makespan-optimal plans. However, it is usually slower then heuristic-based sub-optimal approaches. In this work we propose MacroSatPlan; a SatPlan based planner which exploits macros extracted by Macro-FF and uses a predictive model of the optimal solution length that is constructed by WEKA, a commonly used toolkit of machine learning algorithms.*

Keywords: SAT-based planning; macro-actions; predictive model; machine learning for domain-independent planning.

1 Background

SatPlan [11, 12, 10] is a planner based on the satisfiability approach. It constructs a planning graph [8] up to the first level k that contains all the problem goals. The graph is converted into a set of clauses in conjunctive normal form (CNF). The CNF is solved by a SAT-solver. If the set of clauses is unsatisfiable, the planning graph level k (and the corresponding SAT encoding) is increased and the process repeats. Otherwise the solution of the CNF is translated into a solution for the original planning problem.

MiniSat [5] is the SAT-solver used in this work. The search of MiniSat is based on the DPLL algorithm [2], extended with backtracking by conflict analysis and clause recording [4], and boolean constraint propagation (BCP) [9].The selection of the next unassigned variable to assign is called decision. The effects of the decision are propagated by unit propagation: as soon as 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. The propagation process continue 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 work described in this paper is the result of a joint work with Alfonso E. Gerevini and Alessandro Saetti.

The most critical component of search algorithm is the heuristic for selecting the next variable to assign. MiniSat uses a dynamic variable order that prefers the variables involved in recent conflicts (each variable has an activity indicator associated with it [9]).

MiniSat employs search restarts. A search restart consists of clearing all the decisions made, keeping some of the information gained from the conflict analysis, and then starting again the search.

2 Architecture of MacroSatPlan

The architecture of MacroSatPlan, sketched in Figure 1, consists of three main modules, briefly described below.

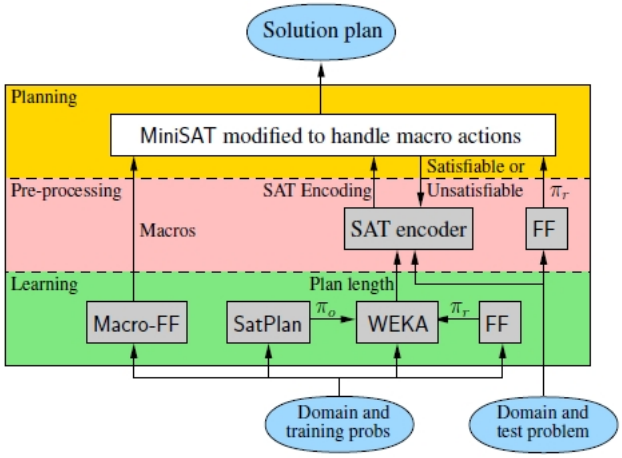

Figure 1: A sketch of MacroSatPlan's architecture. π_o and π_r indicate an optimal plan and a relaxed plan, respectively.

Learning. This module consists of four main components: SatPlan, a modified version of planner FF [1], Macro-FF [13], and the machine learning tool WEKA [7].

FF is a forward search planner which exploits GraphPlan for computing a relaxed plan π_r achieving the problem goals from the successor states. The number of actions in π_r is an estimate of the distance from the successor states to the goal states. We use a revised version of FF that only computes the length of the relaxed plan derived from the initial state of the planning problem, without computing a solution for it.

Macro-FF computes macros by analyzing solutions of a set of problems generated by FF. The macros that appear more frequently and that reduce the required search effort significantly are preferred. Macro-FF orders the extracted macros by evaluating their impact on the run-time performance of FF. MacroSatPlan selects only the best macro computed by this approach.

WEKA is a well-known tool for learning predictive models. Given a set of training problems for domain D, WEKA is used to identify a predictive model of the length of the optimal solution of a generic problem of domain D from (i) the

length of the optimal plan computed by SatPlan, (ii) some pre-identified features of the planning problem, and (iii) the length of the relaxed plan computed by FF. The optimal plan length predicted by WEKA is subsequently used by the SAT encoder of the planning problem as the initial value of the planning horizon. It is important to note that, differently from SatPlan, in our approach the initial horizon can be higher than the optimal plan length.

Preprocessing. For every test problem, FF computes the corresponding relaxed plan. The relaxed plan is used by MacroSatPlan to order and select macros during SAT solving. The more actions of the relaxed plan are included in a macro, the more promising this macro is, considered in terms of its possible presence in a solution (and hence of its usefulness in generating the solution).

SatPlan constructs the planning graph up to the level k estimated by the learned predictive model. If MiniSat solves the problem, the level of the planning graph is (iteratively) decremented in order to find an optimal plan. Otherwise the current planning graph is too short, and the SAT encoder increments k.

Planning. A modified version of MiniSat tries to solve the SAT problem received from the SAT encoder. The new SAT solver exploits macros and the relaxed plan computed by FF to guide the satisfiability search.

2.1 The revised MiniSat algorithm

```
Algorithm 1
Input: A CNF formula encoding a planning problem
Output: SAT or UNSAT
v ← null;
while TRUE do
    propagate(v);
    if not conflict then
        if all variables assigned then
            return SAT;
        else
            if v corresponds to an action then
                res ← propagateNoop(v);
                if res = SAT or res = UNSAT then
                    return res;
            v ← selectVariableFromMacros(Ω,M);
            if v = null then
                v ← selectVariable(Ω);
            Ω ← Ω - v;
    else
        analyze conflicting clause;
        if top level conflict then
            return UNSAT;
        else
            backtracking;
```

Figure 2: The modified MiniSat algorithm. M represents the set of all computed macros. Underlined lines indicate the new parts of the algorithm

The original MiniSat algorithm has been modified for taking advantage of macros. Algorithm 1 is an overview of the modified search procedure. First it tries to select and assign variables belonging to macros. If these choices fail, the original MiniSat heuristic decides the next variable.

Algorithm 2
Input: The set of all computed macros M; the set of unassigned variables Ω; the current variable assignment $\mathcal{W}$.
Output: next variable to assign **or** null
$MF \leftarrow \{m \in M \mid \nexists v \in m, \cdot \mathcal{W} \models (v = False)\}$
$MT \leftarrow \{m \in M \mid \forall v \in m, \cdot \mathcal{W} \models (v = True)\}$
$MS \leftarrow \{m \in M \setminus MF \cup MT \mid \forall v \in m, \nexists m' \in MT \cdot v \in m'\}$
if $\mid MS \mid = 0$ **then**
  return null
$m \leftarrow select(MS)$
$v \leftarrow first(m \cap \Omega)$
return v

Figure 3: The *selectVariableFromMacros* procedure. The *select* procedure is called to order macros and returns the most promising one. Function *first* extracts the unassigned variable that encodes the action at the earlier time step.

The procedure *selectVariableFromMacros*, in Algorithm 2, selects the next variable to decide by analyzing macros. It operates on M, the set of all computed macros, and defines three subsets: MS, MT and MF. MF contains all macros with at least a variable assigned as false; MT contains only macros with all the variables assigned as true (we call them "completely assigned macros"); MS (Macros Selected) contains "activable" macros: macro-actions in which there exists at least one not yet assigned variable.

MS does not include macros with a variable assigned as false. This is to avoid promoting variables in macros that, given the current variable assignment, it is likely that will not appear in the solution plan. MS even does not contain macros with a variable in another completely assigned macro, since we observed that, for several domains, two macro-actions including at least one common action do not happen simultaneously in a solution plan.

If the MS set is empty, the procedure returns null. That is the procedure cannot find a variable to assign. Otherwise, the *select* function is called to order macros and returns the most promising one. It orders macros by (i) number of actions that appear in the relaxed plan, (ii) the ratio between the number of variables assigned as true and the cardinality of the macro, (iii) the sum of the activity values of variables, (iv) the time step of the first action. If none of the ordering criteria returns a single macro, a random macro from the last set will be returned. From the returned macro, the *selectVariableFromMacros* procedure extracts the unassigned variable that encodes the action at the earlier time step, by function *first*.

MacroSatPlan tries to assign Noops everytime it assigns an action. Only Noops of goals' facts are considered. The *propagateNoop* procedure tries to decide variables corresponding to Noops that are encoded in time steps subsequent to the last assigned action.

Further, MacroSatPlan uses the search restarts to switch between the original MiniSat algorithm and the modified one. This policy is intended to combine both the strategies.

3 Conclusions and Future Work

In this paper, we have presented an approach to learning macros and a predictive model for improving SAT planning. Preliminary experimental study shows that

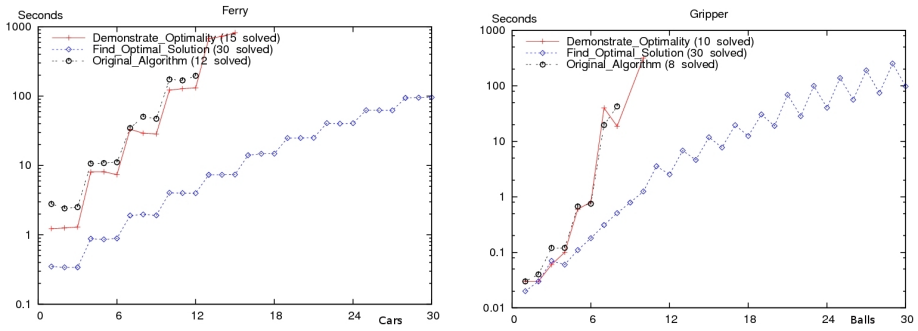

Figure 4: CPU time for Ferry and Gripper

this knowledge can be useful for an enhanced SAT solver. The experimental analysis uses a collection of problems in the well-known domains `Gripper` and `Ferry`.

Figures 4 shows the results of MacroSatPlan for finding the optimal solution (*Find_Optimal_Solution*) and demonstrating its optimality (*Demonstrate_Optimality*) versus the original SatPlan (*Original_Solver*).

Concerning the results of MacroSatPlan, we observe that MacroSatPlan always performs better than the original SatPlan for finding the optimal solution. Furthermore it is generally faster for demonstration of optimality and it is able to demonstrate the optimality of more solutions than SatPlan.

Finally, the most expensive step in the SAT-based planning is the demonstration of optimality. MacroSatPlan finds quickly the optimal solution and uses most of the CPU time for demonstrating its optimality. Original SatPlan does not demonstrate the solution optimality because it generates encodings from a proved lower bound of the optimal plan length.

Future works include additional experiments and the integration of Wizard [3] as another system for learning macro-actions.

References

1. Hoffmann, J., Nebel, B.: The FF planning System: fast plan generation through heuristic search. J. of Artificial Intelligence Research, (2001)
2. Davis, M. and Logemann, G. and Loveland, D.: A machine program for theorem-proving. ACM, (1962)
3. M. A. H. Newton and J. Levine and M. Fox and D. Long: Learning Macro-Actions for Arbitrary Planners and Domains. Proceedings of the 17th ICAPS (2007)
4. Silva, Joao P. Marques and Sakallah, Karem A.: GRASP a new search algorithm for satisfiability. Proceedings of the 1996 ICCD (1996)
5. Niklas, E. and Niklas, S.: An extensible SAT-solver. Proceedings of SAT 2003 (2003)
6. Adele E. Howe, Mark Roberts, Brandon Wilson, Marie desJardins: What Makes Planners Predictable?. Proceedings of the 18th ICAPS (2008)
7. Witten, I. H. and Frank, E., Data Mining: Practical Machine Learning Tools and Techniques (2005)
8. Blum, A. and Furst, M., Fast planning through planning graph analysis. Proceedings of the 14th IJCAI (1995)
9. Moskewicz, M. W. and Madigan, C. F. and Zhao, Y. and Zhang, L. and Malik, S., Chaff: Engineering an Efficient SAT Solver. Proceedings of the 38th DAC (2001)
10. Kautz, H. and Selman, B. and Hoffmann, J., SatPlan: Planning as Satisfiability. Abstracts of the 5th IPC (2006)
11. Kautz, H. and Selman, B. and Hoffmann, J., Planning as Satisfiability. Proceedings of 10th ECAI (1992)
12. Kautz, H. and Selman, B. and Hoffmann, J., Unifying SAT-based and Graph-based Planning. Proceedings of the 16th IJCAI (1999)
13. Botea, A. Muller, M. and Schaeffer, J., Fast Planning with Iterative Macros. Proceedings of the 20th IJCAI (2007)