The Freelunch Planning System Entering IPC 2018

Tomáš Balyo
Karlsruhe Institute of Technology
Karlsruhe, Germany
biotomas@gmail.com

Stephan Gocht
KTH Royal Institute of Technology
Stockholm, Sweden
gocht@kth.se

Abstract

The planners FREELUNCH-DOUBLY-RELAXED and FREELUNCH-MADAGASCAR consist of three components: a) an encoding of the planning problem to SAT (FREELUNCH, which also tries to solve the instance before passing it on, or MADAGASCAR) b) the driver for solving the SAT problems incrementally (INCPLAN) and c) a modern incremental SAT solver with inprocessing (LINGELING).

Introduction

The general idea of our planners is to encode the planing problem into SAT and use an of-the-shelf SAT solver to solve it. However, it turns out that lots of problems can already be solved with a simple heuristic search, which is done in FREELUNCH-DOUBLY-RELAXED before passing it to the SAT solver.

To transform the planning problems into SAT, we use MADAGASCAR (Rintanen, Heljanko, and Niemelä 2006) with the $\exists$ encoding in FREELUNCH-MADAGASCAR and in FREELUNCH-DOUBLY-RELAXED we use the *Selective* encoding (Balyo and Barták 2015) which is a heuristic selector that chooses either the *Relaxed Relaxed Exist-Step (RRES)* encoding (Balyo 2013) or the *Reinforced* encoding (Balyo, Barták, and Trunda 2015).

The *double ended incremental encoding* (Gocht and Balyo 2017) is used by the tool INCPLAN to call the SAT solver LINGELING (Biere 2013) incrementally with inprocessing.

Preliminary Definitions

Incremental SAT Solving

A *clause* is a disjunction (OR) of literals, a *literal* is a Boolean variable or its negation and a *Boolean variable* is variable with two possible values (True and False). A *conjunctive normal form* (CNF) *formula* is a conjunction (AND) of clauses. A CNF formula is satisfiable if there is an assignment of truth values to its variables that satisfies at least one literal in each clause of the formula.

The idea of incremental SAT solving is to utilize the effort already spent on a formula to solve a slightly changed but similar formula. The assumption based interface (Eén and Sörensson 2003) has two methods. One adds a clause C and the other solves the formula with additional assumptions in form of a set of literals A:

$$add(C)$$
$$solve(assumptions = A)$$

Note that we will add arbitrary formulas, but they will be transformable to CNF trivially. The method $solve$ determines the satisfiability of the conjunction of all previously added clauses under the condition that all literals in A are true. Note that it is only possible to extend the formula, not to remove parts of the formula. However, this is no restriction. If we want to add a clause C we plan to remove later we add it with an activation literal: Instead of adding C we add $(a \vee C)$. If the clause needs to be active, $\neg a$ is added to the set of assumptions for the solve step. Otherwise, no assumption is added and the solver can always satisfy the clause by assigning True to a.

SAT-Based Planning

A planning problem is to find a plan – a sequence of actions, that transforms the initial state into a goal state. The basic idea of solving planning as SAT (Kautz and Selman 1992) is to express whether a plan of length i exists as a Boolean formula F_i such that if F_i is satisfiable then there is a plan of length i. Additionally, a valid plan must be constructible from a satisfying assignment of F_i. To find a plan the plan encodings $F_0, F_1, \ldots$ are checked until the first satisfiable formula is found, which is called sequential scheduling and used in our planners. There also exist more advanced algorithms to schedule the solving of plan encodings with different makespans (Rintanen, Heljanko, and Niemelä 2006), however these approaches seem to be less beneficial, when incremental SAT solving is used.

Representation of the Plan Encoding

To apply incremental SAT solving it is necessary to break the plan encoding down to its essential parts. While an arbitrary encoding does not necessarily have this structure, all existing encodings already use this structure or are easily expressed within the presented terms.

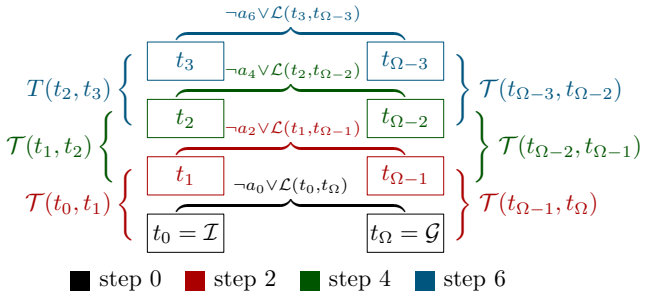

Figure 1: Visualization of the double ended incremental encoding. For clear arrangement, link clauses are only shown for every other step. The colors show which clauses are new in the particular step. All clauses from previous steps are present as well. (Gocht 2017)

The variables of the plan encoding F_i are divided into $i+1$ groups called *time points* with the same number of variables N, $v_k@j$ represents variable k at time point t_j. The clauses of F_i are divided into three groups:

- initial clauses $\mathcal{I}$: satisfied in the initial state t_0
- goal clauses $\mathcal{G}$: satisfied in the goal state t_i
- transition clauses $\mathcal{T}$: satisfied at each pair of consecutive time points $(t_0t_1, t_1t_2, \ldots, t_{i-1}t_i)$

The clauses of $\mathcal{I}, \mathcal{G}$ operate on the variables of one time point and $\mathcal{T}$ operates on the variables of two time points. $\mathcal{T}(j,k)$ indicates that the transition clauses are applied from time point j to time point k and similarly for $\mathcal{I}, \mathcal{G}$. The plan encoding F_i for makespan i can be constructed from these clause sets:

$$F_i = \mathcal{I}(0) \wedge \left(\bigwedge_{k=0}^{i-1} \mathcal{T}(k, k+1) \right) \wedge \mathcal{G}(i)$$

Double Ended Incremental Encoding

Let us also introduce the *double ended incremental encoding* (Gocht and Balyo 2017) as described in (Gocht 2017).

With non-incremental SAT solving the plan encodings are newly generated for each makespan and the SAT solver does not learn anything from previous attempts. With an incremental SAT solver it is possible to append a new time point in each step. The idea of the double ended incremental encoding is to add new states between initial state and goal state, such that clauses can be learned from both. This can be understood as having two stacks: One stack contains the time point with initial clauses at the bottom, the other contains the time point with the goal clauses at the bottom. New time points are pushed alternating to both stacks. The time points at the top of both stacks are linked together with link clauses, such that they represent the same time point, i.e. each variable has the same value in both time points: $\mathcal{L}(j,k) := \wedge_{l=1}^{N} v_l@j \Leftrightarrow v_l@k$. Activation literals ensure that only the latest link is active.

step (0) :
  add $\left(\mathcal{I}(0) \wedge \mathcal{G}'(\Omega) \wedge [\neg a_0 \vee \mathcal{L}(0, \Omega)]\right)$
  solve $(\textit{assumptions} = \{a_0\})$
step $(2k+1)$: add t_{k+1}
  add $\left(\mathcal{T}'(k, k+1) \wedge [\neg a_{2k+1} \vee \mathcal{L}(k+1, \Omega-k)]\right)$
  solve $(\textit{assumptions} = \{a_{2k+1}\})$
step $(2k)$: add $t_{\Omega-k}$
  add $\left(\mathcal{T}'(\Omega-k, \Omega-k+1) \wedge [\neg a_{2k} \vee \mathcal{L}(k, \Omega-k)]\right)$
  solve $(\textit{assumptions} = \{a_{2k}\})$

Note that Ω is neither a precomputed number nor a fixed upper bound but a symbol which always represents the last time point and $\Omega - k$ is the k^{th} time point before the last. In step zero there is no transition between the first time point 0 and the last time point Ω. Therefore, both time points are the same. In step one there is one transition between the first and the last time point. In step two there are two transitions in between and so on. This is visualized in Figure 1.

Implementation

To transform the planning problems into the SAT encoding, i.e. into initial clauses $\mathcal{I}$, transition clauses $\mathcal{T}$ and goal clauses $\mathcal{G}$, we use MADAGASCAR (Rintanen, Heljanko, and Niemelä 2006) with the $\exists$ encoding in FREELUNCH-MADAGASCAR.

In FREELUNCH-DOUBLY-RELAXED we use the *Selective* encoding (Balyo and Barták 2015) which is a heuristic selector that chooses either the *Relaxed Relaxed Exist-Step (RRES)* encoding (Balyo 2013) or the *Reinforced* encoding (Balyo, Barták, and Trunda 2015) based on the relative number of state variable transitions in the problem description. For more details on the selection rule see (Balyo and Barták 2015).

The tool INCPLAN takes care of the double ended incremental encoding and the interaction with the state-of-the-art incremental SAT solver Lingeling (Biere 2013), which is used to solve the formulas. An overview of this general system is shown in Figure 2.

Inprocessing

The used SAT solver Lingeling (Biere 2013) does support inprocessing. To prevent the removal of variables which are necessary to extend the formula we tell the solver to keep all variables which are contained in the link clause, which is active in the current solve step.

Pre-solving with Heuristic Forward Search

In FREELUNCH-DOUBLY-RELAXED we have an initial heuristic search phase. It is run for the first 5 minutes in the Satisfycing Track and 2 minutes in the Agile Track.

Starting with the initial state, the algorithm computes all the applicable actions in the current state that lead to a not yet visited state. For each of these actions a heuristic value is computed representing its supposed usefulness. The action

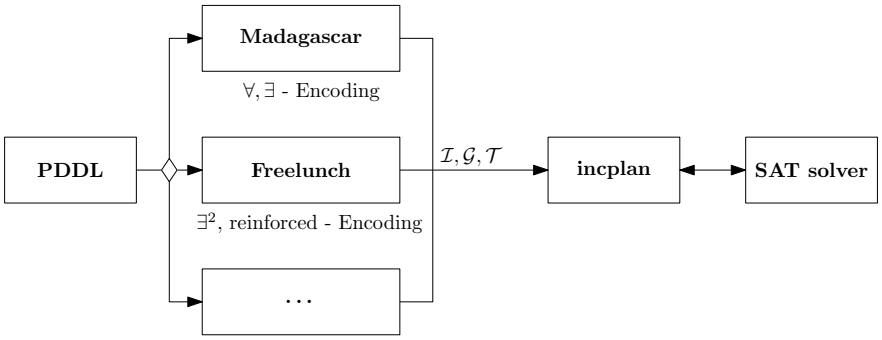

Figure 2: System Overview

with the highest value is selected and applied on the current state. If we get to a state, that each applicable action leads to an already visited state or there is no applicable action, then we backtrack to the previous state.

The heuristic function of action usefulness is very simple and greedy. An action starts with a score of 0. If an effect of the action sets a variable to a goal value while in the current state it has a different value, then the score of the action is increased by 1. On the other hand, if an effect changes the value of a variable which already has a goal value, then the score is decreased by 1. Finally, to break the ties, the score is multiplied by 10 and a random value between 0 and 9 is added to it.

Despite its simplicity, this algorithm can solve around one half of the IPC 2011 benchmark problems very quickly. The downside of the algorithm is that it finds extremely long plans full of redundant actions. For example, for domains such as Elevators and Transport the found plans contain hundreds of thousands of actions while plans found by Fast Downward (Helmert 2006) only have a few hundred actions.

Fortunately, these extremely long plans can be easily reduced to reasonable lengths using post planning optimization techniques. Even the simplest such techniques perform very well on these plans due to their severe redundancy. The post planning optimization algorithm we used is Action Elimination (AE) (Nakhost and Müller 2010; Fink and Yang 1992). AE is a polynomial ($O(|P|^2)$) heuristic algorithm capable of removing redundant (unnecessary) actions from plans. It is not guaranteed to remove all redundant actions (which is an NP-complete problem (Fink and Yang 1992)) and it cannot add/replace actions.

In some cases the plan is very long even after the post planning optimization. The plan may be so long that the widely used plan validation tool val (Howey, Long, and Fox 2004) crashes on it.

Conclusion

In this paper we described the set of tools and the tool-chains we submitted to the IPC 2018 under the name Freelunch. We believe it represents the state-of-the-art in SAT based planning, and we hope it will perform well on the competition's benchmark problems

References

Balyo, T., and Barták, R. 2015. No one satplan encoding to rule them all. In *Eighth Annual Symposium on Combinatorial Search*.

Balyo, T.; Barták, R.; and Trunda, O. 2015. Reinforced encoding for planning as sat. *Acta Polytechnica CTU Proceedings* 2(2):1–7.

Balyo, T. 2013. Relaxing the relaxed exist-step parallel planning semantics. In *ICTAI*, 865–871.

Biere, A. 2013. Lingeling and plingeling home page. `http://fmv.jku.at/lingeling/`.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, 502–518.

Fink, E., and Yang, Q. 1992. Formalizing plan justifications. In *In Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence*, 9–14.

Gocht, S., and Balyo, T. 2017. Accelerating SAT based planning with incremental SAT solving. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, 135–139.

Gocht, S. 2017. Incremental SAT solving for sat based planning.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)* 26:191–246.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, 294–301. IEEE.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *ECAI '92 Proceedings of the 10th European conference on Artificial intelligence*, volume 92, 359–363.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *ICAPS*, 121–128.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12):1031–1080.