Aquaplanning: An Educational Framework for Automated Planning

Tomáš Balyo, Dominik Schreiber, Patrick Hegemann, Jeremias Trautmann
Karlsruhe Institute of Technology, Germany
{tomas.balyo,dominik.schreiber}@kit.edu

Abstract

We present Aquaplanning, a new framework for PDDL and automated planning. The framework has been developed for educational and experimental purposes and may also serve as an efficient and versatile planner with a multitude of different strategies and techniques. For the Sparkle Planning Challenge 2019, we employ a sequential portfolio consisting of two different forward search strategies and a third planning approach based on incremental SAT solving.

Introduction

In the following, we describe the framework Aquaplanning[1] and its configuration that we submit to the Sparkle Planning Challenge 2019. Aquaplanning is a framework written in Java, the name being a recursive acronym for "**A**quaplanning **Qu**ick **A**utomated **Planning**". It has been originally developed for educational purposes, serving as a modular and integrated platform with which students and researchers alike can easily understand, implement and evaluate ideas related to automated planning. Note that Aquaplanning is self-contained: it does not depend on any planning-related frameworks. We focus on keeping the planning instance as an object-oriented and comprehensible structure throughout the entire planning pipeline. As such, modifications and extensions to the employed algorithms are possible in a comparably easy manner.

The document is structured as follows: First, we present the general features and internal workings of Aquaplanning. Then, we describe the specific solving techniques we employ in the Sparkle Planning Challenge 2019.

Features

Aquaplanning fundamentally operates on PDDL as an input format. All PDDL features related to basic classical planning are supported, as well as a range of advanced features such as conditional effects, quantifications, derived predicates (axioms), general disjunctive logic, and numerical conditions. We use an original PDDL parser written on top of the Antlr parser generator (Parr 2013).

[1]https://github.com/domschrei/aquaplanning

Our grounding algorithm features a planning graph traversal (Blum and Furst 1997) in order to find all relevant facts and actions in the provided problem and to simplify the problem accordingly. Actions with disjunctive conditions can be split into multiple actions by constructing the condition's Disjunctive Normal Form (DNF), or they can retain their complex logical structure for the remaining procedure. The grounding is constructed in a way such that operators can have a large number of arguments and still be instantiated in a reasonably efficient way: We order an operator's arguments descendingly by the number of occurrences within the operator's preconditions. Then, using this ordering, we employ a depth-first search on the operator's possible partial assignments to its arguments, immediately discarding any partial assignment which leads to an unfulfilled precondition in the planning graph's current layer.

For planning purposes, we have implemented several notable kinds of planning approaches: in addition to a general interface for heuristic forward search planning as well as a more specialized greedy forward search procedure, we also included a number of Satisfiability-based approaches, encoding the problem into propositional logic and solving it with the help of an incremental SAT solver. For this purpose, either the integrated SAT4j library (Le Berre and Parrain 2010) or any other SAT solver over the incremental SAT interface IPASIR (Balyo et al. 2016) can be used. In the configuration we submit to the challenge, we use MiniSAT (Eén and Sörensson 2003).

Some of the further features of Aquaplanning include a plan validation tool as well as general interfaces for parallel portfolio planning and plan optimization techniques.

Sparkle Planning Challenge 2019

For the upcoming planning challenges, we have constructed a sequential portfolio consisting of three different planners. We explain each of these planners in some more detail in the following sections.

We have found that Aquaplanning's planning approaches have strong capabilities in different kinds of planning domains and, as a consequence, that we can achieve better results with a sequential portfolio than when just employing one of the planners. The approaches are arranged in in the following way: First, we try try to solve the problem using a greedy and very fast planner. If we do not find a plan after

200 seconds, we employ a more informed search strategy for 40 seconds. If this fails as well, we try to resolve the problem with a SAT-based approach, considering that it may have a highly complex logical structure.

Greedy Best-First Forward Search

The following search approach corresponds to the forward search strategy of the Freelunch planner (Balyo and Gocht 2018).

Starting with the initial state, this 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 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 explicitly 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.

Well-Informed Heuristic Search

This approach works similarly to the Greedy Best-First Search approach, but operates in a more informed and less greedy manner. Firstly, we maintain a frontier of all open search nodes instead of just remembering the path leading to the current state. As a result, no explicit backtracking is required and state space can be explored in a more informed manner: In each iteration, the globally most promising search node is visited and expanded. Secondly, as a heuristic, we use the popular concept introduced by the Fast-Forward heuristic (Hoffmann and Nebel 2001) of traversing a delete-relaxed planning graph from the current state, then computing a (relaxed) plan from the layers, and using its cost as a heuristic value. This heuristic is much more costly to compute than the greedy heuristic used in our first approach, but it can serve as a much better guidance through state space on some domains.

Planning as Incremental SAT

Last but not least, we employ a SAT-based approach as first described in (Kautz and Selman 1992): The ground planning problem is encoded into propositional logic for a given number of steps (makespan). Then, a SAT solver attempts to solve the resulting formula. If it succeeds, then a plan can be extracted from the found satisfying assignment to the variables. If it fails, then we extend the formula to admit a higher makespan and retry.

We avoid to re-encode the entire planning problem in each iteration by using incremental SAT solving and encoding the problem in a way such that the formula can be easily extended to any makespan (Gocht and Balyo 2017). We make use of the Exists-step encoding (Rintanen, Heljanko, and Niemelä 2006) where the execution of multiple actions in parallel is possible as long as some possible valid ordering of actions exists. We increase the makespan in steps of five, and the maximum run time for solving each makespan before proceeding to the next makespan is decreased exponentially. This is inspired by Algorithm B from (Rintanen 2004) and aims to benefit from the faster solving times of later satisfiable makespans. By increasing the makespan by five steps at a time, those makespans can be reached faster.

Conclusion

We have presented the educational automated planning framework Aquaplanning. We described its various features and explained our sequential portfolio approach that we employ in the upcoming Sparkle Planning Challenge 2019. We hope that our planning configuration performs well on the challenge's benchmarks.

Acknowledgements

We would like to thank all students participating in the "Automated Planning and Scheduling" lecture at KIT who contributed to Aquaplanning in the form of pull requests, issues, and fruitful discussions.

References

Balyo, T., and Gocht, S. 2018. The freelunch planning system entering IPC 2018. *IPC 2018 planner abstracts*.

Balyo, T.; Biere, A.; Iser, M.; and Sinz, C. 2016. Sat race 2015. *Artificial Intelligence* 241:45–65.

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281–300.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, 502–518. Springer.

Gocht, S., and Balyo, T. 2017. Accelerating sat based planning with incremental sat solving. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *ECAI 92: Tenth European Conference on Artificial Intelligence*, 359–363.

Le Berre, D., and Parrain, A. 2010. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation* 7:59–64.

Parr, T. 2013. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Rintanen, J. 2004. Evaluation strategies for planning as satisfiability. In *ECAI*, volume 16, 682.