PASAR Entering the Sparkle Planning Challenge 2019

Nils Froleyks, Tomáš Balyo, Dominik Schreiber
Karlsruhe Institute of Technology
Karlsruhe, Germany
n.froleyks@gmail.com, {tomas.balyo,dominik.schreiber}@kit.edu

Abstract
We present the planning system PASAR – Planning As Satisfiability with Abstraction Refinement – whose main planning component is based on the principles of SAT-based planning combined with counterexample guided abstraction refinement (CEGAR). As an abstraction of the original problem, we use a simplified encoding where interference between actions is generally allowed. Abstract plans are converted into actual plans where possible or otherwise used as a counterexample to refine the abstraction. In addition, we combine the SAT-based approach with a simple forward state-space search algorithm in order to pre-filter instances where forward search generally tends to perform better than SAT-based planning.

Introduction
Satisfiability (SAT) based planning is one of the well established approaches to automated planning. It is based on the idea of encoding a planning problem instance into a sequence of SAT formulas and then use a SAT solver to solve them. The method was first introduced by Kautz and Selman (Kautz and Selman 1992) and is still very popular and competitive. This is partly due to the rapidly increasing power of SAT solvers, which are getting more efficient year by year and the many new improvements that have been made to the method since its introduction. Some examples of these improvements are new compact and efficient encodings (Huang, Chen, and Zhang 2010; Rintanen, Heljanko, and Niemelä 2006; Robinson et al. 2009; Balyo 2013), better ways of scheduling the SAT solvers (Rintanen, Heljanko, and Niemelä 2006), modifying the SAT solver’s heuristics to be more suitable for solving planning problems (Rintanen, Heljanko, and Niemelä 2006) and – most recently – using incremental SAT solving (Gocht and Balyo 2017).

We introduce a new algorithm named PASAR for SAT based planning by utilizing the counterexample guided abstraction refinement technique (Clarke et al. 2000). The basic idea of PASAR is that we encode the planning problem instance to SAT “incorrectly”, i.e., we leave out some of the clauses, thus obtaining an abstraction of the proper encoding. Then we find a solution to the SAT formula, which can be decoded into a sequence of actions $P'$. If $P'$ is a valid plan, the procedure terminates. Otherwise, we attempt to transform the abstract plan into a valid plan using various techniques from conventional planning as well as a number of custom optimizations. If we fail to compute a valid plan, $P'$ constitutes a counterexample to our abstract encoding and we need to refine it. We refine our abstraction by adding additional clauses (derived from $P'$) to our formula. We repeat the cycle of solving our formula and refining our abstraction until we arrive at a valid plan.

We combine PASAR with a time-capped preliminary stage of forward state-space search in order to “pre-filter” planning instances which tend to be solvable more easily with forward search techniques.

Preliminaries
We give some necessary preliminaries for the techniques we employ in our planner.

SAT-Based Planning
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 $F_i$ is satisfiable if and only if there is a plan of length $i$ or less. 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. The following constraints are encoded:

0. The values of state variables in the first step are consistent with the initial state.
1. The values of state variables in the final step are consistent with the goal conditions.
2. At each step, each state variable assumes exactly one value from its respective domain.
3. If an action is executed at step $t$, then its preconditions are satisfied at step $t$ and its effects are valid at step $t + 1$.
4. If a value of a state variable changes between steps $t$ and $t + 1$, then there must be an action at step $t$ causing this change by one of its effects.
Algorithm 1: PASAR planning procedure

1. Procedure PASAR($\Pi = (X, O, s_0, s_G)$)
2. $k := 1$
3. $F := $ AbstractEncode($\Pi$);
4. while true do
5.   $F_k := $ Instantiate($F, k$);
6.   result := Solve($F_k$);
7.   if result = UNSAT then
8.     $k := $ IncreaseMakespan($k$);
9.   else
10.      $\rho := $ ExtractAbstractPlan($\Pi$);
11.      $\pi := $ RepairPlan($F, \rho$);
12.      if $\pi \neq FAILURE$ then
13.         return $\pi$
14.      end
15.   end
16. end

Algorithm 2: Procedure of converting an abstract plan into an actual plan

1. Procedure RepairPlan($F, \langle s_0, A_0, s_1, A_1, \ldots, s_k \rangle$)
2. $\pi := \langle \rangle$
3. for $i = 0, \ldots, k - 1$ do
4.   if InterferenceGraph($A_i$) is cyclic then
5.     $A'_i := $ Replan($s_i, s_{i+1}$);
6.     if $A'_i = FAILURE$ then
7.       $F := $ RefineAbstraction($F, A_i$);
8.       return FAILURE;
9.     end
10.   else
11.     $\pi = \pi \circ ValidOrdering(A'_i)$
12.   end
13. end
14. return $\pi$;

CEGAR for SAT-based Planning

Our CEGAR-based planning procedure is described in Algorithm 1. We begin to encode the given planning problem into a set of propositional logic rules $F$ that can be instantiated into an actual formula $F_k$ for any number of steps $k$ (makespan). Note that we do not add clauses from rule (5), thus allowing any number of parallel actions as long as the preconditions and effects of each applied action are met.

For increasing values of $k$, we attempt to solve the corresponding formula until the solver reports satisfiability. Now, a sequence $\rho$ of action sets $A_i$ and intermediate states $s_i$ can be extracted from the satisfying assignment. Next, we call the sub-procedure RepairPlan($F, \rho$) as described in Algorithm 2 to check the validity of this abstract plan and, if necessary, try to repair invalid steps in it.

The structure of an abstract plan is illustrated in Fig. 1. For each adjacent pair of states $(s_i, s_{i+1})$ that the solver found, we check if the set of executed actions at step $i$ is consistent. We describe the dependencies between actions in $A_i$ with a simple interference graph where nodes are actions and a directed edge $(a_1, a_2)$ indicates that $a_1$ requires a precondition that is removed by $a_2$ (in other words: $a_1$ must be applied before $a_2$). It follows that there is a valid ordering of actions in $A_i$ if and only if the interference graph is cycle-free. If there is a valid ordering, we can find it by applying a topological sort on the graph. If the graph does contain a cycle, then the involved nodes correspond to the set of conflicting actions.

If the interference graph of $A_i$ contains a cycle, we may attempt to replan this invalid step by starting a separate planning procedure where $s_i$ is the initial state and $s_{i+1}$ is the fully defined goal state. We employ a greedy best-first search approach driven by a Manhattan distance heuristic with ties broken randomly. Note that this planning task is supposed to be comparably easy if the plan is reparable, so we fix a certain small time frame to solve this planning problem. If no solution is found, then we consider the set of interfering actions as a counterexample to a valid plan. We add non-interference clauses from rule (5) for the concerned set of actions to the abstract formula $F$ and report that the previously found abstract plan cannot be repaired. The procedure is then restarted for the same makespan $k$, but with a refined formula $F_k$.

There are two major reasons for specifically omitting the non-interference clauses from the initial encoding. Firstly, non-interference clauses are one of the most expensive parts of a naive SAT encoding of a planning problem. Secondly, the extent to which non-interference clauses are added to the encoding essentially characterizes the parallel action semantics that is realized in the encoding. As we begin without any non-interference clauses, we realize a form of action parallelism that is equivalent to the exists-step semantics (Rintanen, Heljanko, and Niemelä 2006) until we find some invalid, irreparable plan. Afterwards, restrictions are added gradually until a plan is found, resulting in a foreach-step semantics in the worst case where non-interference clauses are added for each pair of conflicting actions.

Note that we have added various optimizations and techniques to PASAR on top of the abstract procedure we just introduced clauses of the form $\preceq(a, b) \lor \neg\preceq(a, b)$ for each pair of actions $(a_1, a_2)$ and the corresponding Boolean variables $(do(a_1), do(a_2))$ which indicate their execution at step $t$.
described, which will be explained in-depth in a separate publication.

Forward Search Pre-Planning

We have found that the “pure” PASAR approach is capable of solving complex instances by exploiting SAT solving as a powerful logical resolution backend. However, sometimes, this effort is poorly invested as some planning domains share a logical structure that is resolvable much more easily by basic forward state-space search algorithms. As a consequence, the planning system which we submit to the Sparkle Planning Challenge 2019 will begin with a greedy best-first search procedure up to a certain time limit (100 seconds), using a Manhattan goal distance heuristic with ties broken randomly. If no plan is found after this limit, the actual PASAR procedure will be employed to resolve the instance by exploiting our planner’s full potential.

Implementation Details

We provide some essential details regarding the implementation of our approach.

We use the grounding of PDDL problem files into SAS+ provided by Fast Downward (Helmert 2006) to translate problem instances into a ground representation that is well-suited for the encoding into propositional logic.

We utilize incremental SAT solving for our planning procedure, i.e., only one single CNF formula is maintained and extended over the course of the entire algorithm. For each call of Instantiate($F,k$) (Algorithm 1, line 5), the abstract clauses $F$ are added as necessary until all constraints for steps 0, 1, . . . , $k$ are encoded. The problem’s goal is encoded as a set of assumptions that is considered only for the upcoming Solve($F_k$) call and dropped afterwards. This way, clauses never need to be removed from the formula even when the makespan is extended.

We use IPASIR, a generic interface for incremental SAT solving (Balyo et al. 2016). As IPASIR can be used together with any popular SAT solver, we chose Glucose (Audemard and Simon 2009), a highly popular and award-winning SAT solver, as our solving backend.

Conclusion

We presented our planning system PASAR that utilizes the paradigm of Counterexample-based Abstraction Refinement in combination with SAT-based planning. Employing simple forward search techniques enables us to “pre-filter” the planning instances which are solvable more easily by conventional forward search techniques. We hope that our planner will perform well in the Sparkle Planning Challenge 2019.

References


