Computing Multiple PDR Steps in a Single SAT Call and a PDR Comparison to Madagascar with Completeness Thresholds

Marshall Clifton, Charles Gretton
Canberra, ACT, Australia 0200
{marshall.clifton, charles.gretton}@anu.edu.au

Abstract

Property Directed Reachability (PDR) is a sound and complete SAT-based procedure for classical planning problems. As with all SAT-based planners, PDR operates according to the notion of a plan *step*. Existing works limit what can occur in a plan step to what is achievable instantaneously by executing planning actions that do not interfere or conflict.

We have developed two new PDR planners each of which explores a broadening of what can occur in a plan step, by allowing a sequence of actions to be executed.

We also report on experimentation using the ramp-up SAT-based planner *Madagascar*, and in particular focus on its performance as a complete procedure given a *completeness threshold* made available by recent developments in the SAT-based planning setting.

This is a short version of a longer paper submitted to the ICAPS 2021 Doctoral Consortium (DC)[1].

Introduction

The classical propositional deterministic planning problem is that of determining whether or not a goal condition is reachable from an initial state by executing a finite sequence of actions. The problem is represented succinctly in a propositionally factored form, such as STRIPS or PDDL (McDermott et al. 1998). A broad family of planning procedures solve the problem via a series of related queries to a SAT(isfiablity) decision procedure. A much studied approach involves bounding the problem with a horizon, and solving the bounded problem with a general purpose SAT solver (Kautz and Selman 1992) *Property Directed Reachability* (PDR), is a recently proposed sound and complete procedure in this setting (Bradley 2011; Suda 2014). It tracks and maintains reachability information for each step away from the goal in a compact efficient logical representation.

In this paper, we compare a PDR planner to a SAT-based planner using a given set of fixed horizons. Separately, we also evaluate variations of PDR that increase the workload of each individual SAT solver invocation by giving it a larger portion of the problem. This is done with the aim that by exposing more of the problem, the SAT solver will be able to increase the efficiency of the overall procedure.

[1]Please contact the first author for a copy of the longer DC submission

Planning via Boolean SAT

The basics of Boolean SAT solving are here assumed knowledge. Where ϕ is a formula in propositional logic, we write $\vdash \phi$ to indicate that ϕ is satisfiable. Where α is an assignment, we write $\alpha \vDash \phi$ to indicate that α is a satisfying assignment of ϕ. We define the fixed-horizon classical planning problem by providing a direct $\forall$-step description of that problem as a formula in Boolean logic (Rintanen, Heljanko, and Niemel 2006). Let X be the finite set of all state-characterising facts. To describe the planning problem, for each $x \in X$ we require a SAT proposition $x@t$ meaning: x is *true* at the state encountered at step t. Where $x@t$ is a proposition, we write $\overline{x@t}$ for its negation, meaning x is *false* at step t. Similarly, for each action a, we require a SAT proposition $a@t$ meaning a is executed at step t, we write $\overline{a@t}$ for its negation, meaning a is not executed at step t.

A planning state, s, at time step t is described by a complete assignment to the facts X, and can be represented by a conjunction of literals $\mathsf{Lits}(s,t)$. Let I be the set of facts that are initially true in the planning problem, then we have that the initial state, s^I, is represented by the conjunct:

$$\mathsf{Lits}(s^I,0) \equiv \bigwedge_{x\in I} x@0 \wedge \bigwedge_{x\in X\setminus I} \overline{x@0} \tag{1}$$

In words, facts in I are exclusively true initially (i.e. at step 0). Where G is the planning goal—i.e. set of facts that must be achieved—then for the direct encoding of the fixed horizon, $h \geq 0$, planning problem in SAT, we have the constraint:

$$\bigwedge_{g\in G} g@h \tag{2}$$

All goal facts in G are true of the state encountered in the final step h.

Let A be the finite set of planning actions. For each action $a \in A$ we have: (i) a set of facts, $\mathrm{pre}(a) \subseteq X$, called "preconditions", that must be satisfied at step t for a to be executable at t, (ii) the set of facts that are true, $\mathrm{add}(a) \subseteq X$, called "add effects", in the successor state supposing a is executed, and (iii) the set of facts, $\mathrm{del}(a) \subseteq X$, called "delete effects", that are false in the successor. Formally, we have the following clauses:

$$\begin{aligned}
&\forall a \in A.\ \forall x \in \text{pre}(a).\ \ t \in 1 \ldots h.\ \ \overline{a@(t-1)} \vee x@(t-1)\\
&\forall a \in A.\ \forall x \in \text{add}(a).\ \ t \in 1 \ldots h.\ \ \overline{a@(t-1)} \vee x@t\\
&\forall a \in A.\ \forall x \in \text{del}(a).\ \ t \in 1 \ldots h.\ \ \overline{a@(t-1)} \vee \overline{x@t}
\end{aligned} \tag{3}$$

Lastly we require frame axioms that explain how the truth values of state facts change between successive timesteps. To ensure that all state transitions are explained by action execution, we have the following:

$$\begin{aligned}
&\forall x \in X.t \in 1 \ldots h.\\
&\quad x@(t-1) \vee \overline{x@t} \vee \textstyle\bigvee_{a \text{ s.t. } x \in \text{add}(a)} a@(t-1)\\
&\forall x \in X.t \in 1 \ldots h.\\
&\quad \overline{x@(t-1)} \vee x@t \vee \textstyle\bigvee_{a \text{ s.t. } x \in \text{del}(a)} a@(t-1)
\end{aligned} \tag{4}$$

The conjunction of $\mathsf{Lits}(s^I, 0)$ with axioms from Equations 2, 3 and 4 gives the CNF formula corresponding to the "∀-step" encoding of the fixed horizon planning problem. As is custom, we admit multiple actions being set true at a step, provided those do not *conflict*—i.e. an action deletes the precondition of another—or *interfere*—i.e. one deletes/adds an add/delete effect of the other.

∃-step semantics, described originally in (Dimopoulos, Nebel, and Koehler 1997) and applied to planning by (Rintanen 2012) often allow multiple actions which conflict to be scheduled together, if there exists an ordering of the actions which permits a valid plan. This expands what can be scheduled in a timestep. Some of our experiments are with systems that employ this semantics.

A *completeness threshold* is a horizon such that if the bounded planning problem at all bounds less than or equal to this horizon do not admit a solution, then the original problem does not have a solution (Biere et al. 2003).

Backward Search: Property Directed Reachability

The PDR approach to SAT-based planning is to search backwards from the goal G. For increasing distances $h \in \{0, .., \hat{h}\}$, this approach iteratively discovers a series of CNF formulae, $\mathcal{L}_i$ for $i \leq h$ over symbols in X. We call these formulae the *layer* information. For $x \in X$ we again find the concept of $x@t$, the proposition stating x is true at step t, to be useful. $\mathcal{L}_i@t$ denotes the rewriting of $\mathcal{L}_i$ by replacing propositional symbols x with their timed counterpart $x@t$. Each $\mathcal{L}_i$ defines an overapproximation of states that are i steps from the goal. In other words, for every planning problem state s that can reach a goal state within i steps, the backwards search maintains the invariant condition $\vdash \mathsf{Lits}(s, i) \wedge \mathcal{L}_i@i$. For $i > 0$ every $\mathcal{L}_i$ initially corresponds to the vacuously satisfied empty CNF formula – i.e. the loosest possible overapproximation. For $h = 0$, $\mathcal{L}_0$ corresponds to the conjunction in Eq 2 omitting the timing suffix on symbols – i.e. any occurrence of $x@h$ in Eq 2 is rewritten as x in $\mathcal{L}_0$. PDR begins by checking that the problem is nontrivial – i.e. if $\nvdash \mathsf{Lits}(s^I, 0) \wedge \mathcal{L}_0$, then the problem is nontrivial because it can only be solved by a nonempty plan. Then PDR proceeds iteratively, at the h^{th} iteration processing a family of timestamped states, represented by their conjunctions $\mathsf{Lits}(s, i)$ where $i \leq h$. Each $\mathsf{Lits}(s, i)$ identifies with the assumption that from s the goal can be achieved within i steps. Q is a container, comprised of a stack for each layer index. When a state index pair (s,i) is added, state s is added to stack i. When an item is retrieved from the container, it is retrieved from the stack with the lowest index. Timestamped states are processed with respect to Q, which at the beginning of the h^{th} iteration contains only $\mathsf{Lits}(s^I, h)$. An iteration is completed when Q is emptied, and while Q remains nonempty its elements are removed, one at a time, and evaluated as follows.

- **CASE 1:** An element $\mathsf{Lits}(s, i)$ is removed from Q and there exists an assignment α such that:
$$\alpha \vDash \mathsf{Lits}(s, i) \wedge T@i \wedge \mathcal{L}_{i-1}@(i+1) \tag{5}$$
Above, $T@i$ corresponds to the conjunction of axioms of the form given in Eqs 3 and 4, fixing $t = i$. In other words, $T@i$ is the formula describing what state transitions are possible from the state indicated by $\mathsf{Lits}(s, i)$. Writing $\alpha \lfloor Y$ for the assignment projected to variables in the set Y, the assignment $\alpha \lfloor X@(i+1)$ describes a successor state s' satisfying the overapproximation $\mathcal{L}_{i-1}$. Lastly in this case, PDR adds $\mathsf{Lits}(s', i-1)$ and $\mathsf{Lits}(s, i)$ to Q.
- **CASE 2:** No α exists satisfying Eq 5. A subformula r of $\mathsf{Lits}(s, i)$ known as the *reason* is then derived so that when r is substituted for $\mathsf{Lits}(s, i)$ in Eq 5 that formula remains unsatisfiable. Descriptions of PDR offer some flexibility regarding exactly how r is computed. In the worst case $r = \mathsf{Lits}(s, i)$, and in practice we attempt to minimise the number of terms in r. The negated timeless reason $\bigvee_{l \in \mathsf{Lits}(r,i)} \bar{l}$, corresponds to the disjunctive clause containing the timeless negation of all literals in $\mathsf{Lits}(r, i)$ – i.e. if $x@i$ occurs in $\mathsf{Lits}(r, i)$, the corresponding timeless l would be x, and $\overline{x}$ would appear as a literal. The CNF formulae $\forall j \in \{0, .., i\}$. $\mathcal{L}_j$ are all updated to include this negated timeless variant of the reason. Intuitively, here we have discovered that s is not i steps from a goal state, and have added a (minimum) *nogood* to $\mathcal{L}_i$ and all lower layers to reflect this. It remains to determine whether or not s is $i+1$ steps away from the goal, and therefore if $i < h$ we add $(s, i+1)$ to Q.

In our implementation, r is found iteratively, by repeatedly finding smaller subformulae of $\mathsf{Lits}(s, i)$.

In case a term of the form $\mathsf{Lits}(s, 0)$ is discovered the algorithm terminates as the planning problem at hand is proved satisfiable. Although peripheral to our current study, with some auxiliary accounting it is straightforward at this point to extract a plan (Suda 2014). Once Q is empty and the h^{th} iteration is complete, PDR adds clauses to the $\mathcal{L}$ CNFs so that there is no consecutive $\mathcal{L}_i$ and $\mathcal{L}_{i-1}$ where:
$$\nexists c \in \mathcal{L}_i \text{ s.t. } \nvdash \mathsf{Lits}(\bar{c}, i) \wedge T@i \wedge \mathcal{L}_{i-1}@(i+1) \tag{6}$$
Above, $\mathsf{Lits}(\bar{c}, i)$ is the negation of the disjunctive clause c at step i. If at any time two consecutive formulae, $\mathcal{L}_i$ and $\mathcal{L}_{i-1}$, become equivalent, then PDR has proved that no plan exists and the search can be aborted. In practice, equivalence is performed by a syntactic check – i.e. treating the CNF as a set of clauses, and checking that the set of clauses in two consecutive $\mathcal{L}$ CNFs are identical.

Using Transition Macros

We propose a variation of PDR which we later show can decrease planner time, by processing multiple planning steps in a single SAT call. This is done with the aim, that by exposing more of the problem to the SAT solver, the solver can arrive to conclusions faster than if small components are given piece by piece. Processing multiple steps involves having multiple corresponding *intermediate* states represented in the SAT instance. These intermediate states are not exposed to the PDR process. Where $\mathcal{F}$ is the number of planning steps to be taken together, we replace $T@i \wedge \mathcal{L}_{i-1}@(i+1)$ in Eq 5 and Eq 6 with the multi-step *macro*:

$$\left(\bigwedge_{n=0}^{n=\mathcal{F}-1} T@(i+n)\right) \wedge \mathcal{L}_{i-1}@(i+\mathcal{F}) \tag{7}$$

The successor state s' is then $\alpha \lfloor X@(i+\mathcal{F})$. We refer to this procedure as *PDR-M*.

Interleaved Layering

We propose another similar variation, which uses the layer information $\mathcal{L}$ to constrain the intermediate states. For this we replace $T@i \wedge \mathcal{L}_{i-1}@(i+1)$ in Eq 5 and Eq 6 with the multi-step variation with *interleaved* layer information:

$$\left(\bigwedge_{n=0}^{n=\mathcal{B}-1} T@(i+n) \wedge \mathcal{L}_{i-n-1}@(i+n+1)\right) \tag{8}$$

Where $\mathcal{B} = min(\mathcal{F}, i)$. $\mathcal{B}$ plays a similar role in Eq 8 as $\mathcal{F}$ does in Eq 7, but allows for the case where i is too small, and there are not enough layers. Unlike the macro approach above, because the intermediate states are constrained by the layer component of the formula, they can be extracted and added to Q.

The check to see if no plan exists also needs to be changed. Unlike traditional PDR which requires 2 successive layers to be equivalent, here, $\mathcal{F}+1$ successive layers need to be equivalent. As before, this is done symbolically. Because of this, a higher h is required to prove no plan exists, which can be a bottleneck. We refer to this procedure as *PDR-IL*.

Experimental Results

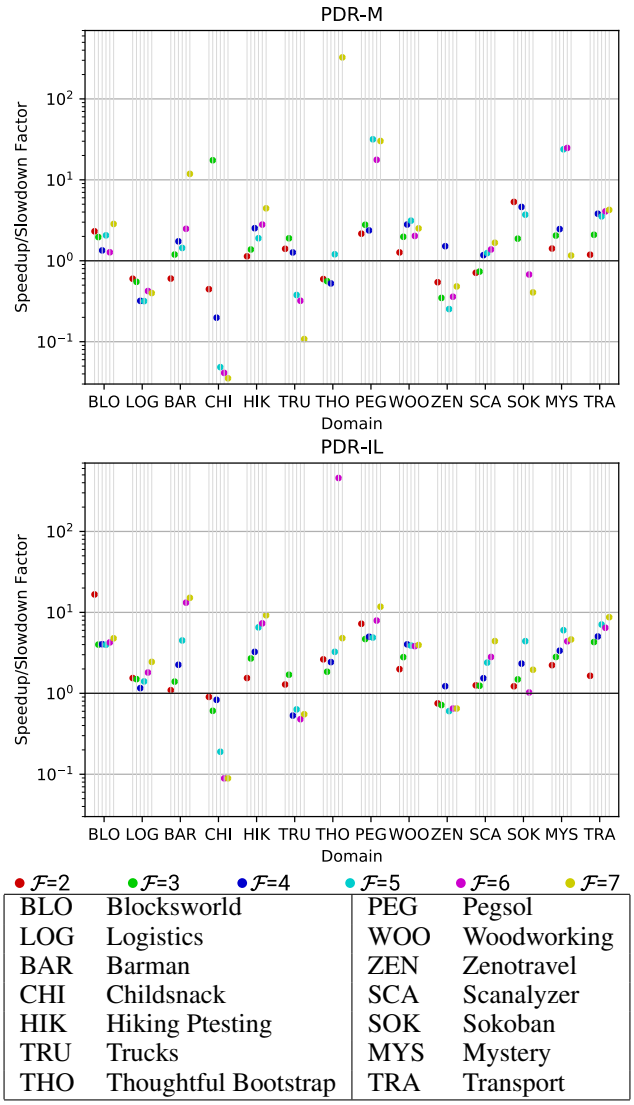
All experiments were performed on an Intel Xeon Gold 6134 CPU@3.2GHz with 20GB Memory.

PDR, PDR-M and PDR-IL

The effectiveness of using the PDR, PDR-M and PDR-IL is evaluated. We implemented PDR using Lingeling as the SAT solver (Biere 2017). We include *invariants*, constraints important for performance, from the plangraph as specified in (Robinson et al. 2009).

The runtime performance of all PDR planners is dominated by the time spent in the underlying satisfiability procedure, which is what is measured and reported in this section.

Our implementation of PDR was tested on a subset of benchmark planning problems from the International Planning Competitions up to 2014. When $\mathcal{F} = 1$, PDR-M, PDR-IL and traditional PDR are equivalent, we refer to this solver as the *baseline*.



BLO	Blocksworld	PEG	Pegsol
LOG	Logistics	WOO	Woodworking
BAR	Barman	ZEN	Zenotravel
CHI	Childsnack	SCA	Scanalyzer
HIK	Hiking Ptesting	SOK	Sokoban
TRU	Trucks	MYS	Mystery
THO	Thoughtful Bootstrap	TRA	Transport

Figure 1: Average speedup/slowdown factors for each $\mathcal{F}$ in $\{1, .., 7\}$. Vertical lines left to right indicate values for $\mathcal{F}$, 2 through 7.

We observed that the runtime performance of planners could vary significantly for different values of $\mathcal{F}$.

If any planner in this comparison cannot solve a problem, for any value of $\mathcal{F}$, then we exclude that problem from reporting.

We have run each PDR procedure under investigation on each problem, using values of $\mathcal{F} \in \{1, .., 7\}$. For each run we measured the total time spent by the planner in the underlying Boolean satisfiable procedure – i.e. here that is Lingeling. To report our experimental findings here, we have summarised the collected runtime data by reporting the average speedup/slowdown factor relative to the baseline, plotted in Figure 1. This is the average ratio of time spent relative to the baseline—i.e. a higher value means the variation was slower. The runtime factor of the baseline procedure—i.e. taking $\mathcal{F} = 1$—is uniformly 1, and the runtime factors

Table 1: Standard Deviation of Speedup/Slowdown Factors for PDR-M

Domain	PDR-M						PDR-IL					
(#Tasks)	$\mathcal{F}$=2	3	4	5	6	7	2	3	4	5	6	7
BLO (88)	4.85	3.98	0.99	3.87	1.46	13.19	82.51	3.29	3.28	2.77	3.1	3.30
LOG (74)	0.34	0.58	0.22	0.21	0.24	0.23	0.67	0.81	0.74	0.84	2.4	3.91
BAR (14)	0.39	1.09	2.78	1.31	2.3	13.52	0.88	0.96	1.71	4.21	29.14	20.87
CHI (4)	0.45	34.50	0.36	0.05	0.05	0.04	1.46	0.71	1.53	0.22	0.09	0.09
HIK (9)	0.53	0.52	1.93	1.02	2.76	2.44	0.46	1.13	1	5.02	5.58	6.16
TRU (2)	0.71	0.31	0.9	0.24	0.26	0.02	0.78	0.56	0.21	0.19	0.02	0.18
THO (2)	0.24	0.11	0.17	0.97	3422.9	457.56	1.97	0.2	0.34	0.64	642.35	0.37
PEG (9)	1.38	2.23	1.99	50.25	34.22	72.78	13.24	5.31	4.37	3.67	4.58	9.31
WOO (20)	0.42	0.78	1.33	1.75	0.95	1.18	0.55	1.03	1.64	1.63	1.59	1.63
ZEN (6)	0.54	0.26	1.15	0.26	0.35	0.58	0.82	0.7	1.12	0.52	0.55	0.5
SCA (3)	0.41	0.33	0.67	0.6	0.92	1.01	1.14	1.02	1.50	2.82	3.53	6.34
SOK (8)	8.73	2.57	11.61	9.54	0.94	0.53	1.94	2.1	2.59	9.83	1.05	3.08
MYS (7)	1.51	1.65	2.11	58.94	61.67	0.86	1.99	2.48	2.71	7.59	4.06	4.29
TRA (5)	0.49	1.15	2.31	1.87	3.85	5.47	0.79	4.01	2.37	5.83	4.50	10.26

plotted for the other planners are the average values across the domain relative to this baseline. The standard deviations between speedup/slowdown factors are shown in table 1.

It is more common to be able to speedup planning using PDR-M than when using PDR-IL. Notably in the Logistics domain, for PDR-M there is a consistent speedup for all $\mathcal{F}$ values tested, especially for $\mathcal{F} = 4$, which has a standard deviation of 0.22.

PDR v.s. Ramp-Up Given Upper Bound

PDR is capable of proving a problem unsat. Traditional SAT-based procedures can only prove unsat for a given horizon bound. Here we contrast the runtime of fixed horizon methods with PDR for unsat problems. Madagascar is a state of the art SAT planner which is used in this comparison (Rintanen 2012). As this comparison is independent of the previously mentioned work regarding PDR-M and PDR-IL, Madagascar will be compared to standard PDR. The implementation of PDR, PDRPlan was used (Suda 2014). A subset of the 2012 IPC unsatisfiable track was used as the benchmarks. Madagascar was used to solve each problem with various horizons, including when available, a completeness threshold from the state of the art generator *PlanBound* (Abdulaziz 2019). When run with the horizons 10, 20, 40 and 80, the $\exists$-step semantics was used.

Table 2 shows the number of problems that were solved in the allowed time, broken down by solver, horizon and domain. Here, for Madagascar, solved means that the solver was able to disprove the existence of a plan at the given horizon.

The experimental datapoints reported for PDR thus far correspond to runtimes of a complete planning procedure. Madagascar is complete given a horizon bound equal to the completeness thresholds provided by PlanBound. For each problem, PlanBound was run with a 1800s timeout. Here, for cases where a bound $<$10,000 is found (22 problems), we run Madagascar with a timeout of 220 hours. Madagascar is able to prove the instances unsat in 5 of those 22 problems, and in such cases is always able to do so using Unit Propagation exclusively – i.e. no decisions are made.

Table 2: Number of instances solved by PDRPlan, and Madagascar with various horizons.

Domain (#Tasks)	PDR	Madagascar			
		h=10	20	40	80
Bag Gripper (25)	0	4	4	4	1
Bag Transport (29)	1	25	2	0	0
Bottleneck (25)	16	25	21	16	14
Cave Diving (25)	5	25	7	5	4
Chessboard Pebbling (23)	3	23	23	0	0
Diagnosis (20)	15	20	20	19	15
Document Transfer (20)	4	20	19	11	9
Over Nomystery (24)	12	18	11	2	2
Over Rovers (20)	16	20	9	6	5
Pegsol (24)	14	20	8	4	4
Pegsol Row 5 (15)	4	15	4	3	3
Sliding Tiles (20)	0	20	20	0	0

Conclusion

We have compared PDR with a traditional SAT planner over various horizons, including where possible a completeness threshold. We evaluated these over a set of unsatisfiable benchmark planning problems. We found using PDR on unsatisfiable problems far more effective than traditional SAT planning with a completeness threshold.

We also introduced two parameterizable variations of PDR which solve multiple planning steps in a single SAT call. These approaches, under various parameters, were evaluated over a set of planning problems. We found that using our PDR variations, we can achieve a substantial (greater than 10 times) speedup over the baseline in some cases. However, the speedup gained is very sensitive to the domain chosen and the parameters used.

References

Abdulaziz, M. 2019. Plan-Length Bounds: Beyond 1-Way Dependency. *AAAI Conference on Artificial Intelligence* 33: 7502–7510.

Biere, A. 2017. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In Balyo, T.; Heule, M.; and Järvisalo, M., eds., *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, 14–15. University of Helsinki.

Biere, A.; Cimatti, A.; Clarke, E. M.; Strichman, O.; and Zhu, Y. 2003. Bounded model checking. *Advances in Computers* 58: 117–148.

Bradley, A. R. 2011. SAT-based model checking without unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, 70–87. Springer.

Dimopoulos, Y.; Nebel, B.; and Koehler, J. 1997. Encoding planning problems in nonmonotonic logic programs. In *Recent Advances in AI Planning*, volume 1348, 169–181. Berlin, Heidelberg: Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

Kautz, H. A.; and Selman, B. 1992. Planning as Satisfiability. In *ECAI*, volume 92, 359–363.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL: The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial intelligence* 193: 45–86. Publisher: Elsevier.

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

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2009. SAT-based parallel planning using a split representation of actions. In *Nineteenth International Conference on Automated Planning and Scheduling*.

Suda, M. 2014. Property directed reachability for automated planning. *Journal of Artificial Intelligence Research* 50: 265–319.