AxSAT – Bringing Axioms to SAT Planning

Gregor Behnke[1][0000−0002−1445−9934], David Speck[2][0000−0002−5493−7363], and Daniel Gnad[3,4][0000−0001−7434−2669]

[1] ILLC, Universiteit van Amsterdam, The Netherlands
[2] University of Basel, Switzerland
[3] Heidelberg University, Germany
[4] Linköping University, Sweden
g.behnke@uva.nl davidjakob.speck@unibas.ch daniel.gnad@uni-heidelberg.de

Abstract. Planning as SAT is, in addition to explicit and symbolic search, one of the main approaches for solving planning problems. Such planners proved very successful, especially in combinatorially complex domains. SAT-based planning has to date focused on the core formalisms of planning. Notably, there is no SAT-based planner that supports axioms and derived predicates. In this paper, we present our new planner AxSAT that supports axioms as well as conditional effects. Furthermore, we show how to allow for action parallelism using the ∃-step encoding in the presence of axioms. Our empirical evaluation shows that AxSAT performs favorably compared to state-of-the-art approaches for satisficing classical planning with axioms, and provides complementary capabilities.

1 Introduction

A central aspect of intelligent decision-making is reasoning before acting, which involves determining a sequence of actions to transform the current state into a desired one. Such reasoning problems are typically described as classical planning tasks in dedicated modeling languages such as PDDL [25]. While PDDL is rich in modeling features allowing for concise encoding of planning tasks, many of them are not supported by modern planners. One such feature is the use of axioms, which provide a means to specify a background theory to derive certain facts about the world in the current state, and thus often allow for natural, elegant and efficient modeling and problem specification [43, 21, 26, 15, 40]. Importantly, axioms enable the modeling of complex action preconditions and goal conditions. Despite the fact that axioms are an essential modeling feature, i.e., they cannot simply be compiled away [43], their expressive power poses a challenge to support them. This is especially true for planners based on the heuristic-search paradigm. It is challenging to design heuristics (goal-distance estimators) that are both informative and fast, while taking into account the background theory [42]. In particular, admissible heuristics used in optimal planning are complex to adapt [21], but for inadmissible heuristics such as the FF heuristic [18] it poses a challenge, too. Thus, axioms are often treated naively, making the incorporation of heuristic estimates difficult.

In this paper, we focus on planning as satisfiability [23, 32]. We show how planning tasks with axioms can be encoded as a propositional satisfiability (SAT) problem with a fixed (parallel) plan length bound. Our approach generalizes existing SAT planning encodings, such as the sequential encoding, $\forall$-step encodings [24], and the $\exists$-step encoding [10, 32].

While our approach is the first to model planning axioms as boolean satisfiability, there have been related approaches. Bofill et al. [6] considered a domain-specific SAT implementation of axioms for a puzzle game, which is not easily generalizable to domain-independent planning. Other works have introduced axiom support for compilations of planning to integer programming and answer-set programming [26, 9]. None of the approaches supports parallelism, the encodings are purely sequential. Speck et al. [39, 42] showed how to support axioms in symbolic search. Their dominant approach is to precompute the formulas under which derived facts become true, in order to replace those facts in the conditions with the computed formula. While this approach is similar to ours, there are critical differences. On the theoretical side, based on our encodings, SAT solvers can infer the same formulas, but it is done lazily, only when necessary to show satisfiability instead of a precomputation. On the practical side, symbolic search is mainly used for optimal planning, while SAT-based planning has its merits mainly in satisficing planning. As we will show empirically, these two approaches complement each other well: the symbolic search approach of Speck et al. [39, 42] is strong in finding optimal plans, while our SAT-based approach, AxSAT, shows strong performance in satisficing planning. Overall, AxSAT compares favorably to state-of-the-art approaches for satisficing planning with axioms such as the well-known heuristic search planner LAMA [29].

2 Preliminaries

In this section, we provide the necessary background for our work by introducing classical planning with axioms and planning as satisfiability.

2.1 FDR Planning

We consider planning problems with conditional effects given in Finite Domain Representation (FDR) [17], of which we describe the base formalism first. States are described via a set of state variables $\mathcal{V}$, each $v \in \mathcal{V}$ having its own finite domain D_v. A partial state s is any partial function from variables $\mathcal{V}$ to values. We write $V(s)$ to denote the set of variables for which s is defined. A state is a function mapping each variable v to a value D_v, i.e., $s : \mathcal{V} \mapsto D_v$. An action is a pair $a = \langle p, e \rangle$, where p is a partial state and e is a set of expressions $l \triangleright r$ where l is a partial state and r is a fact of the form $v = o$ for a variable $v \in \mathcal{V}$ and a value $o \in D_v$. Given an action a, we write $pre(a)$ to refer to its precondition p and $\mathit{eff}(a)$ for its effects e. An action a is applicable in a state s, if s agrees with p for every variable for which p is defined, i.e., if $p \subseteq s$. Let e be a set of conditional effects. The firing conditional effects $f(e, s)$ in a

state s are defined as $f(e,s) = \{v = o \mid l \triangleright v = o \in e, l \subseteq s\}$. We write $V(f(e,s))$ for the variables mentioned in the firing conditional effects. The set of all effects of conditional effects $\alpha(e)$ is defined as $\alpha(e) = \{v = o \mid l \triangleright v = o \in e\}$. Applying a set of conditional effects e to a state s leads to the state $\gamma(s,e) = \{(v, s(v)) \mid v \notin V(f(e,s))\} \cup f(e,s)$. Technically, this definition can lead to conflicts if two firing conditional effects set the same variable to different values. We assume by definition that this is not possible. Applying a in s leads to a successor state $\gamma(s,a) = \gamma(s,e)$ if applicable, otherwise to $\bot$. For a sequence of actions $\pi = (a_1, \ldots, a_n)$, we define its application to a state s inductively as $\gamma(s,(a_1)) = \gamma(s,a_1)$ and $\gamma(s,(a_1,a_2,\ldots,a_n)) = \gamma(\gamma(s,a_1),(a_2,\ldots,a_n))$. An FDR planning problem Π is then specified by its variables $\mathcal{V}$, actions $\mathcal{A}$, initial state s_I, and its goal s_G, which is a partial state. A plan that solves this FDR problem is any sequence of actions π such that $s_G \subseteq \gamma(s_I, \pi)$.

2.2 Derived Predicates and Axioms

In this paper, we use an extension to pure FDR planning: derived predicates [43, 17]. Derived predicates are logical atoms and thus can only have two values: true and false. To keep our notation clean, we make an explicit distinction between the original finite domain variables and the derived predicates. To represent them, we denote the set of derived predicates as $\mathcal{D}$. A partial state s is now a pair $\langle F, \delta \rangle$, where F is a partial function from the FDR variables $\mathcal{V}$ to values and $\delta \subseteq \mathcal{D}$ is a set of derived predicates that is true in s. States are partial states for which F is a total function $F : \mathcal{V} \mapsto D_v$. The truth value of derived predicates is instead determined using a logic-program-style set of rules, called axioms. An axiom for the derived predicate $d \in \mathcal{D}$ is a formula of the form $\bigwedge_i (v_i = o_i) \wedge \bigwedge_j d_j \wedge \bigwedge_k \neg d_k \rightarrow d$. It is an implication whose left-hand side can contain positive conditions on the FDR variables (negation is not allowed here), and both positive and negative condition on derived predicates. The right-hand side is a single positive derived predicate d which is defined to be true whenever the left-hand conditions are satisfied. For an axiom ax, we will write $\ell(ax)$ to denote the set of literals on its left-hand side and $r(ax)$ for the derived predicate on its right-hand side. Furthermore, we write $\ell^f(ax)$ to denote the set of conditions on FDR variable on the left-hand side and $\ell^+(ax)$ and $\ell^-(ax)$ for the positively and negatively occurring derived predicates respectively. Thus, an axiom can be written as $\bigwedge_{(v_i = o_i) \in \ell^f(ax)} (v_i = o_i) \wedge \bigwedge_{d_j \in \ell^+(ax)} d_j \wedge \bigwedge_{d_k \in \ell^-(ax)} \neg d_k \rightarrow r(ax)$. An axiom ax is applicable in a state $\langle F, \delta \rangle$ iff (1) $\forall v = o \in \ell(ax) : F(v) = o$, (2) $\forall d \in \ell(ax) : d \in \delta$, and (3) $\forall \neg d \in \ell(ax) : d \notin \delta$. Applying an axiom ax in a state $\langle F, \delta \rangle$ yields the state $\langle F, \delta \cup \{r(ax)\} \rangle$. Let A be the set of all axioms of an FDR problem. Given a pure FDR state $F : \mathcal{V} \mapsto D_v$, we define the saturated state $sat(F)$ of F under A as a set-inclusion-minimal fixpoint $\langle F, \delta \rangle$ of applying all axioms $ax \in A$. Intuitively, this means starting with the state $\langle F, \emptyset \rangle$, applying all axioms until no further new axiom is applicable. This definition causes issues if $\ell(ax)$ contains negative literals, as the fixpoint might not be unique. We restrict the set of allowed axioms to Stratified Axiom Programs, for which the fixpoint is unique [33]. As a tool for this definition and our later work, we define the truth

dependency graph (TDG) of the derived predicates as follows: $G = (\mathcal{D}, E)$, where $\mathcal{D}$ is the set of derived predicates, and we have an edge $(d_i, d_j) \in E$ if there is an axiom $ax \in A$ where $d_i \in \ell(ax)$ or $\neg d_i \in \ell(ax)$ and $d_j = r(ax)$. Next, we consider the strongly connected components (SCCs) of the TDG. If there is an axiom ax where $\neg d_i \in \ell(ax)$ and $d_j = r(ax)$ while d_i and d_j are in the same SCC, then evaluating axioms might have non-unique fixpoints. If there is no such axiom, then the set of axioms is a Stratified Axiom Program. In this paper, we only consider such planning problems – notably all domains of the International Planning Competition[5] containing axioms are of this type.

Lastly, we define the application of actions in this setting. An action o comprises four elements $\langle p, p^+_{\mathcal{D}}, p^-_{\mathcal{D}}, e\rangle$ where p is a partial pure FDR state, $p^+_{\mathcal{D}}, p^-_{\mathcal{D}} \subseteq \mathcal{D}$, and e is a set of conditional effects $(l, l^+_{\mathcal{D}}, l^-_{\mathcal{D}}) \triangleright (v = o)$ where l is a partial pure FDR state and $l^+_{\mathcal{D}}, l^-_{\mathcal{D}} \subseteq \mathcal{D}$. An action $\langle p, p^+_{\mathcal{D}}, p^-_{\mathcal{D}}, e\rangle$ is applicable in a state $\langle F, \delta\rangle$ iff $p \subseteq F$, $p^+_{\mathcal{D}} \subseteq \delta$, and $p^-_{\mathcal{D}} \cap \delta = \emptyset$. The firing conditional effects $f(e, s)$ in a state $\langle F, \delta\rangle$ are defined as $f(e, s) = \{v = o \mid l, l^+_{\mathcal{D}}, l^-_{\mathcal{D}} \triangleright v = o \in e, l \subseteq F, l^+_{\mathcal{D}} \subseteq \delta, l^-_{\mathcal{D}} \cap \delta = \emptyset\}$. Applying a set of conditional effects e to a state $s = \langle F, \delta\rangle$ leads to the state $\gamma(s, e) = sat(\{(v, F(v)) \mid v \notin V(f(e, s))\} \cup f(e, s))$, i.e., we apply all FDR effects and then saturate the axioms. Note that thereby the planner is *forced* to evaluate all axioms in every state. Application of sequences of actions is again defined inductively. The initial state is given as a pure FDR state for which the axioms need to be saturated. The goal comprises $p, p^+_{\mathcal{D}}, p^-_{\mathcal{D}}$ mirroring the preconditions of actions. The task is again to find a sequence of actions that, if applied in the initial state, leads to a goal state. Such a plan is optimal if there is no shorter sequence of actions that also solve the problem.

2.3 SAT Planning

One means to solve planning problems is via reduction to Boolean satisfiability (SAT). Given a length bound N, we generate a propositional formula ϕ_N that is satisfiable iff there is a plan of at most N actions. This SAT formula ϕ_N represents the plan as a sequence of N time steps. To obtain a complete SAT-based planner, we then iterate over the length bound N until a solution has been found[6]. The basic encoding has been proposed by Kautz and Selman [23] – but is formulated for STRIPS planning only. There is a dedicated encoding of FDR planning into SAT [19], but its size is cubic in the size of variable domains. We instead opted to adapting the STRIPS-style encoding to FDR planning directly. Here, we present the encoding for FDR planning without derived predicates. The SAT formula comprises two types of variables $(v = o)@t$ indicating that variable v has value o at time t and $a@t$ indicating that action a is executed at time t. Whenever the time step is clear from context, typically if the encoding is repeated for every time step, we will omit the $@t$ to improve readability. For every variable $v \in \mathcal{V}$, we assert that exactly one of its values is true – which can be split into at least and at most one value. The at-least-one constraint is a simple

[5] see https://www.icaps-conference.org/competitions/.

[6] There are more intelligent options than pure iteration, which we discuss in Section 6.

disjunction $\bigvee_{o \in D_v}(v = o)$. For encoding the at-most-one constraint $amo(X)$ for a set of decision variables X, multiple formulae have been proposed [11, 38]. We use the binomial encoding if $|X| \leq 256$ and the binary encoding otherwise. Preconditions and effects of actions are asserted by implications: $\forall (v = o) \in pre(a) : a@n \rightarrow (v = o)@n$ and

$$a@n \wedge \bigwedge_{v'=o' \in l}(v' = o')@n \rightarrow (v = o)@(n+1) \quad \forall l \triangleright (v = o) \in \mathit{eff}(a)$$

To ensure that the truth value of unchanged variables remains the same, we introduce the following frame axiom:

$$\neg(v = o)@n \wedge (v = o)@(n+1) \rightarrow \bigvee_{\substack{a \in \mathcal{A} \\ l \triangleright v=o \in \mathit{eff}(a)}} (a@n \wedge \bigwedge_{v'=o' \in l} (v' = o')@n)$$

In case of conditional effects with non-empty condition, additional decision variables via Tseitin encoding [44] are introduced to compactly encode the right-hand side of the implication. Furthermore, we need to assert that the initial state and goal are met via $(v = o)@1$ for $s_I(v) = o$ and $(v = o)@(N+1)$ for $s_G(v) = o$. Lastly, for the base encoding, we have to assert that at most one action is performed at every time, i.e., $amo(\{a@n \mid a \in \mathcal{A}\})$.

Some extensions of SAT-encodings to more expressive planning formalisms have been proposed in the past. These include lifted planning [20] and hierarchical planning [3, 4, 35, 2, 27]. Similarly, extensions of SAT solving have been used to solve problems in more expressive planning formalisms, notably quantified boolean formulae solving (QBF) for lifted planning [37] and SAT modulo theory solving for temporal and numeric planning [31, 8, 34, 7].

2.4 Parallelism in SAT Planning

On its own, the presented base encoding is not competitive with the state of the art in search-based planning. To achieve this, action parallelism has been added to the encoding. Instead of restricting to a single action to be executed at every time step, one allows for multiple actions – as long as their execution does not interfere. The most common definition of "execution non-interference" is the $\exists$-step semantics. There are more relaxed versions (the relaxed-$\exists$-step [45] and relaxed-relaxed-$\exists$-step semantics [1]), which we do not consider in this paper.

Definition 1 (Based on Rintanen [32]). *Let $E \subseteq \mathcal{A}$ be a set of actions. We call this set $\exists$-step executable in the state s, if there is a total ordering $\langle a_1, \ldots, a_n \rangle$ of the actions in E such that $(a_1, \ldots, a_n)$ is applicable in s.*

Determining for such a set E whether it is $\exists$-step executable is NP-complete [32, Thm. 2.22]. As such, SAT encodings assert and check only a more relaxed version of this $\exists$-step criterion. The main idea is that we fix a total order of all actions $\mathcal{A}$, i.e., $\vec{\mathcal{A}} = \langle a_1, \ldots, a_n \rangle$ and assert that if a set of actions E is

executed, then it is executed precisely in this order. This then makes testing for restricted $\exists$-step parallelism trivial, as we solely have to check the executability of subsequences $\vec{\mathcal{A}}|_E$. This restriction is however not enough to ensure that encoding the executability of the sequence is easy. The $\exists$-step encoding further asserts:

Definition 2 (Based on Rintanen [32]). *Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a sequence of actions. We call π $\exists$-step encodable in s, iff (1) all a_i are applicable in s, (2) $\forall i < j \forall v \in V(pre(a_j)) \cap V(\alpha(\mathit{eff}(a_i))) : \alpha(\mathit{eff}(a_i))(v) = pre(a_j)(v)$, i.e., no action deletes something needed by a later action, and (3) $\forall i < j \forall l \triangleright (v' = o') \in \mathit{eff}(a_j) \forall v \in V(l) \cap V(\alpha(\mathit{eff}(a_i))) : \alpha(\mathit{eff}(a_i))(v) = l(v)$, i.e., no action deletes the condition of a later effect.*

To simplify the SAT encoding, conditions (2) and (3) are independent of the actual state. By construction if $\langle a_1, \ldots, a_n \rangle$ is $\exists$-step encodable, it is $\exists$-step executable.

We say that an action a_i disables an action a_j iff the order $\langle a_i, a_j \rangle$ would violate either condition (2) or (3). Operationally, disabling of actions can equivalently be determined using the sets of deleted and needed facts for each action. For each fact $v = o$, the set of actions needing this fact – where $v = o$ is either a precondition or a condition of a conditional effect – is defined as $nee(v = o) := \{a \in \mathcal{A} \mid v = o \in pre(a)\} \cup \{a \in \mathcal{A} \mid l \triangleright r \in \mathit{eff}(a), v = o \in l\}$. The set of deleting actions is defined as $del(v = o) := \{a \in \mathcal{A} \mid l \triangleright v = o' \in \mathit{eff}(a), o \neq o'\}$. Then, an action a_i disables a_j iff there is a fact $v = o$ such that $a_i \in del(v = o)$ and $a_j \in nee(v = o)$. The disabling graph (DG) encodes the disabling relation between actions. The DG's nodes are the actions and there is an edge from a_i to a_j iff (i) both are potentially executable in the same state and (ii) a_i disables a_j. Determining condition (i) is PSPACE-complete. To approximate, two actions a_i and a_j are presumed to be executable in the same state if there is no $v \in \mathcal{V}$ such that $pre(a_i)(v) \neq pre(a_j)(v)$ while both are defined. To encode the $\exists$-step semantics into a SAT formula, we select an ordering of the actions $\vec{\mathcal{A}}$. If there is an edge (a_i, a_j) in the DG, we prefer to have a_j before a_i in $\vec{\mathcal{A}}$, in which case the constraints of Def. 2 are automatically satisfied. If this is not possible due to cyclic dependencies, we choose any order of the actions involved in the cycle. Formally, we select $\vec{\mathcal{A}}$ as the reverse topological order of the condensation[7] of the DG. For every strongly connected component of the DG, we need to encode the constraints of Def. 2 in the SAT formula. This is done using so-called *chains* [32]. One chain is generated for every fact $(v = o)$ ensuring that after a deleting action $a \in del(v = o)$ no needing action $a' \in nee(v = o)$ in $\vec{\mathcal{A}}$ is executed. We omit details of the encoding and refer the reader to Rintanen [32, Sec. 3.2.2.].

3 Evaluating Axioms in SAT

Our objective is to extend the capabilities of SAT-based planning to being able to handle derived predicates and axioms efficiently. This comprises two challenges:

[7] To condensate a graph, we replace each of its strongly connected components with a single node.

firstly, we need to be able to evaluate axioms correctly and efficiently inside the SAT formula; secondly, we need to integrate axioms into the reasoning of the $\exists$-step encoding. In this section, we start by addressing the first challenge.

3.1 Using the Truth Dependency Graph

To obtain a compact encoding, we utilize the structural information contained in the TDG. We observe that if there is no path from d_i to d_j in the TDG, then determining the truth value of d_j is independent of the truth value of d_i. Consequently, we consider the strongly connected components (SCCs) and their dependency structure. Notably, we determine a topological ordering of the TDG's SCCs as $\vec{S} = \langle S_1, \ldots, S_n \rangle$ where the S_i are sets of derived predicates inside an SCC and $i < j$ implies that there is no path in the TDG from a predicate in S_j to a predicate in S_i. For this ordering, the evaluation of the truth value of any $d \in S_i$ depends only on the truth value of derived predicates $d' \in S_j$ with $j \leq i$. Since by the same argument, the truth value of d' does not depend on d iff $j < i$, we can evaluate the truth values of the derived predicates in each SCC one after another. Notably the truth of $d \in S_i$ depends only the finally determined truth values of all $d' \in S_j$ with $j < i$ and not on any intermediate value. As such we can introduce decision variables $d^*@n$ representing the value of the derived predicate d at time n. The encoding of an SCC S_i will only refer to variables $d^*@n$ for $d \in S_j$ with $j < i$ or additional variables internal to the SCC. Actions a at time n with a precondition on d are similarly encoded with the implication $a@n \rightarrow d^*@n$. To evaluate the derived predicates, we can now handle the SCCs of the TDG independently of each other.

3.2 Base Encoding

Consider a SCC S of the TDG. In an naive attempt, one could try to simply convert each axiom into one clause of the SAT formula per time step. This leads to problems due to self-supporting cycles in the axioms, i.e., we might not find the smallest fixpoint of axiom evaluation. As an example, suppose there are only two axioms: $d_1 \rightarrow d_2$ and $d_2 \rightarrow d_1$. In any state, none of these axioms fire, so both d_1 and d_2 are false. If we directly add the two axioms as clauses into a SAT formula, there are two valuation that satisfy them: one where both d_1 and d_2 are false and one where both are true.

To only allow the first valuation, we need to disallow such self-supporting cycles. There are multiple methods of doing so. We opted for the conceptually simplest one: to evaluate the axioms in steps or layers, where derived predicates in layer ℓ can only become true based on the truth values in layer $\ell - 1$. Such an encoding avoids cycles by construction, but can have substantial size for large SCCs S. Several other methods for breaking cycles have been proposed in the literature (e.g. [28, 13, 12]). We did not experiment with these more complex encodings so far, as the simpler layer-based encoding was sufficient for the benchmarks we considered. We consider this to be future work.

Given the number of layers L, we introduce SAT variables $d_i^\ell @n$ for $0 \leq \ell \leq L$ that indicate that $d_i \in S$ is true in layer ℓ for every time step n. Since the encoding of axioms is identical for every time step, we omit the indication of the time step from now on. For layer 0, we statically assert all derived predicates to false (i.e. $\neg d_i^0$). For every layer, we need to force the derived predicate d_i to true if the conditions of an axiom ax that has d_i as a consequent are met. Consider an axiom ax of the form $\bigwedge_{(v_i=o_i)\in\ell^f(ax)}(v_i = o_i) \wedge \bigwedge_{d_j\in\ell^+(ax)} d_j \wedge \bigwedge_{d_k\in\ell^-(ax)} \neg d_k \rightarrow d$. The left-hand side of ax contains two types of derived predicates: those that are in the current SCC S and those that are part of earlier SCCs. For the latter, we can use the decision variables d_j containing their already determined value. Only for the variable in the same SCC, we need to consider the layering structure. We thus introduce the following clause $\bigwedge_{(v_i=o_i)\in\ell^f(ax)}(v_i = o_i) \wedge \bigwedge_{d_j\in\ell^+(ax)\cap S} d_j^{\ell-1} \wedge \bigwedge_{d_k\in\ell^-(ax)\cap S} \neg d_k^{\ell-1} \wedge \bigwedge_{d_j\in\ell^+(ax)\setminus S} d_j^* \wedge \bigwedge_{d_k\in\ell^-(ax)\setminus S} \neg d_k^* \rightarrow d_j^\ell$. Next, we need to ensure that the derived predicate is true only if there is an axiom that makes it true. So we need to add a clause of the form $d_j^\ell \rightarrow \bigvee_{ax} \big(\bigwedge_{(v_i=o_i)\in\ell^f(ax)}(v_i = o_i) \wedge \bigwedge_{d_j\in\ell^+(ax)\cap S} d_j^{\ell-1} \wedge \bigwedge_{d_k\in\ell^-(ax)\cap S} \neg d_k^{\ell-1} \wedge \bigwedge_{d_j\in\ell^+(ax)\setminus S} d_j^* \wedge \bigwedge_{d_k\in\ell^-(ax)\setminus S} \neg d_k^* \big)$. Since this formula is not in CNF, we introduce intermediate Tseitin variables [44] for the right-hand side of the disjunction, as we do for conditional effects in the base encoding. We do not have to encode a reason for the axiom to stay false – as we force it to be true if one of the axioms fire and axioms cannot force a derived predicate to become false.

Next, we need to determine how many layers to use. Theoretically, it is sufficient to choose the longest simple path inside the SCC S. However, computing it is NP-complete, and even the approximation is hard within a constant factor [22]. Thus, we simply use the size of S as an upper approximation. This however leads to an encoding with at least $\mathcal{O}(|S|^2)$ many clauses. Lastly, we identify d_i^L with the decision variables d_i^* containing the value determined for d_i.

3.3 Efficient Handling Special SCCs Structures

In practice, that is, in our experiments detailed in Section 6, we often find that some SCCs of the TDG have a special structure. The presented base encoding for axioms does not take advantage of these structures. Therefore, we propose two cases for which axioms can be encoded more efficiently.

Implicational SCCs Some domains have derived predicates that express a notion of reachability, e.g., the fact of x being above y, directly or indirectly. The axioms related to these derived predicates have a very simple form: $d_i \rightarrow d_j$ – denoting that if d_i is true, then d_j must be true. Formally, we distinguish two types of axioms: axioms ax are external for the SCC S iff $\ell(ax) \cap S = \emptyset$, i.e., if no derived predicate of the SCC occurs as an antecedent in ax. Otherwise, they are internal. An SCC S is implicational, if all internal axioms ax are of the form $d_i \rightarrow d_j$.

For implicational SCCs the truth of the derived predicates depends solely on the initial values of the derived predicates as set by external axioms. The key

insight is that external axioms need only to be evaluated once – as the truth of their antecedents is fixed and cannot change if the SCCs are processed in topological order. Next, due to the SCC being implicational, a derived predicate d_j is true in the final layer if and only if there is a chain of internal axioms $d_i \to d_{i'} \to \cdots \to d_j$ such that d_i is set to true by an external axiom.

We can exploit this to reduce the number of layers required to fully evaluate the axioms to two. For the first layer, we generate the clauses determining the truth of the d_i^1 variables restricting only the external axioms using the same encoding as in the general case above. Next, we will determine for every $d_i \in S$ the set of derived predicates that will turn true if d_i is true. This can be done by depth-first search over the internal axioms and results in the set $R(d_i)$. Similarly, we compute $R^{-1}(d_i) = \{d_j \mid d_i \in R(d_j)\}$ – the set of all derived predicates d_j that if true force d_i to be true. Given an implicational SCC S, we add the following clauses, evaluating the axioms deriving predicates in S in layer two: $\forall d_i \in S \forall d_j \in R(d_i) : d_i^1 \to d_j^2$ and $\forall d_i \in S : d_i^2 \to \bigvee_{d_j \in R^{-1}(d_i)} d_j^1$. As before we identify d_i^2 with the final value d_i^*.

One Variable Dependent SCC Implicational SCCs do occur in planning problems, but are relatively restrictive. We can extend the same idea to a slightly more complex type of SCCs. In these SCCs, we allow for one additional literal on the left-hand side of axioms. That is, we consider SCCs S in which all internal axioms are of the form $v_k = o_k \wedge d_i \to d_j$, with $v_k \in \mathcal{V}$, $o_k \in D_{v_k}$ and $d_i, d_j \in S$.

We use a core property of the FDR formalism: a variable $v \in \mathcal{V}$ can have only a single value in each state. For this, we require that the additional literal in all internal axioms of the SCC refers to the same variable v, i.e., that the axioms are of the form $v = o_k \wedge d_i \to d_j$.[8] We can now split the axioms into sets A_{o_k} – depending on the variables value. If in the current state $v = o_\ell$, we know that all axioms A_{o_k} for $k \neq \ell$ will never fire as their antecedents are false, while the axioms in A_{o_ℓ} effectively turn into implicational axioms. We can use a similar encoding as for the implicational SCCs while adding a guard for the value of the variable. For this, we compute $R_{o_k}(d_i)$ as the set of all derived predicates reachable from d_i using only axioms in A_{o_k} and $R_{o_k}^{-1}(d_i) = \{d_j \mid d_i \in R_{o_k}(d_j)\}$. Given a one variable dependent SCC S, we add $\forall d_i \in S \forall o_k \in D_v \forall d_j \in R_{o_k}(d_i)$ the clause $v = o_k \wedge d_i^1 \to d_j^2$ and $\forall d_i \in S \forall o_k \in D_v$ the frame axiom: $d_i^2 \wedge v = o_k \to \bigvee_{d_j \in R_{o_k}^{-1}(d_i)} d_j^1$. This frame axiom is sufficient as we know that the variable v will have exactly one value.

4 Parallelism and Axioms

The encoding of axioms described in the previous section solely extends SAT planning to allow for evaluating axioms after **one** action has been executed.

[8] We also allow for purely implicational axioms as well as dependency on a single other derived predicate from a different SCC. They can be viewed as a set of axioms $v = o \wedge d_i \to d_j$ for all values $o \in D_v$.

However, a critical element of efficiency for modern SAT-based planners is the ability to find parallel plans, i.e., plans in which at every time step multiple actions can be executed in parallel.

For adapting the $\exists$-step encoding to axioms, we need to change (1) the construction of the DG and (2) potentially adapt the *chain* clauses. To construct the DG, we need to determine, for every fact f (either an FDR fact or a derived predicate literal), the set $nee(f) \subseteq \mathcal{A}$ of actions that need f, and the set $del(f) \subseteq \mathcal{A}$ of actions that delete it. For $nee(f)$ nothing changes – an action a needs a derived predicate d or its negation $\neg d$ iff it occurs in its precondition or a conditional effect condition.

For $del(f)$, the effects of an action can, via axioms, have an indirect effect on derived predicates. We need to determine which derived predicates could be made true or false by an action. Determining this exactly is NP-complete, as we can use FDR variables not mentioned by a to choose a valuation and the axioms to evaluate a SAT formula. Thus, we approximate. We maintain four sets T^a_{def}, T^a_{pos}, F^a_{def}, and F^a_{pos} – sets of facts that the action a could **pos**sibly $(T^a_{\text{pos}}, F^a_{\text{pos}})$ or **def**initively $(T^a_{\text{def}}, F^a_{\text{def}})$ make **T**rue or **F**alse, respectively. Whenever we add a derived predicate d to any of these sets, we add $\neg d$ to the respective negated set. We first compute definitive effects. For FDR variables, we add for every effect $v = o_i$ of action a, $v = o_i$ to T^a_{def} and $v = o_j$ for $o_i \neq o_j$ to F^a_{def}. Next we handle definitive effects on derived predicates. For every axiom ax, where $\ell(ax) \subseteq T^a_{\text{def}}$, we add $r(ax)$ to T^a_{def}. For every derived predicate d for which all axioms ax that derive d, i.e., all axioms with $d = r(ax)$ have $\ell(ax) \cap F^a_{\text{def}} \neq \emptyset$, we add d to F^a_{def}. We repeat this until convergence.

We proceed to compute possible effects by tracing possible changes through the axioms. Initially, we set $T^a_{\text{pos}} := T^a_{\text{def}}$ and $F^a_{\text{pos}} := F^a_{\text{def}}$. If there is an axiom ax so that $T^a_{\text{pos}} \cap \ell(ax) \neq \emptyset$ and $F^a_{\text{def}} \cap \ell(ax) = \emptyset$ – that is at least one of its conditions can possibly turn true, but none of the conditions definitely turns false – the axiom might be newly applicable and might turn $r(ax)$ true. We thus add $r(ax)$ to T^a_{pos}. We cannot take into account whether any condition is actually false in a given state as we need to derive $del(\cdot)$ state independently. It cannot be the case that $r(ax) \in F^a_{\text{def}}$ as it can only be added if $F^a_{\text{def}} \cap \ell(ax) \neq \emptyset$. Similarly, if there is an axiom ax so that $F^a_{\text{pos}} \cap \ell(ax) \neq \emptyset$ and $r(ax) \notin T^a_{\text{def}}$, i.e., for which one condition might turn false and the right-hand-side is not known to definitely turn true via another axiom, we add $r(ax)$ to F^a_{pos}.

Given these sets, we can approximate that an action a potentially makes all literals in F^a_{pos} false, i.e., we add a to all $del(d)$ for $d \in F^a_{\text{pos}}$. From this point on, we use the standard definition of the DG, i.e., an action a_i disables a_j if they have non-contradictory preconditions (here we don't consider axioms at all) and if $\exists v = o : a_i \in del(v = o) \wedge a_j \in nee(v = o)$. The generation of the $\exists$-step's chain constraints then proceeds as normal – treating derived predicates d as if they were regular facts using the updated definition of $del()$. As F^a_{pos} over-approximates which literals over derived predicates can become false, the conditions (2) and (3) of Def. 2 are satisfied with respect to the derived predicates. This ensures correctness of the encoding.

5 Handling Large Disabling Graphs

This section describes how we handle large DGs in practice.

In some domains, the sets $del(f)$ and $nee(f)$ can be large for some facts f (more than 1.000 actions, mostly for derived predicates). Here, the DG can contain up to $|del(f)| \cdot |nee(f)|$ edges, which can exhaust the planner's memory. In these cases, often at most one of the affected actions can be executed at any time anyway, as all actions disable each other, making any explicit reasoning about them in the DG useless. Hence, we commit to executing at most one of these actions and will always execute it as the last action of a time step. Then, we accumulate these actions in a set $\mathfrak{L}$ of last actions.

We use a threshold T[9] to determine when the disabling edges for a single fact should not be generated. If $|del(f)| \cdot |nee(f)| > T$, we add all actions in $del(f)$ to the set of last actions $\mathfrak{L}$ and do not generate any edges for them. When computing the action ordering $\vec{\mathcal{A}}$ for the $\exists$-step encoding, we ignore actions in $\mathfrak{L}$ and place them last in the ordering. For the $\exists$-step semantics, we do not generate any chain for f and add an at-most-one constraint over all action variables $a@n \in \mathfrak{L}$ for all time steps n.

To prove soundness, consider a case in which a set of actions is selected that is not executable. This set must contain exactly one action a^* out of $\mathfrak{L}$, otherwise it is executable by virtue of the correctness of the $\exists$-step encoding. Of this set, a^* will be executed last. So this sequence can only be non-executable if a^*'s preconditions are not met. This would only be the case, if there is an action deleting a fact that a^* has as precondition. If this is a fact for which a chain was generated, a^* cannot be selected to be executed. Else, all deleters of this fact are part of $\mathfrak{L}$. As a^* was selected, no other $a \in \mathfrak{L}$ can be chosen for execution, and thus a^*'s precondition could not have been deleted. For completeness, note that it is possible to select only a single action $a \in \mathfrak{L}$ and no action $a \in \mathcal{A} \setminus \mathfrak{L}$ and vice versa.

6 Empirical Evaluation

We implemented our planner AxSAT, using the described encoding, on top of Fast Downward (FD) in version 23.06 [16]. Our experiments were conducted on a cluster of Intel Xeon Gold 6130 CPUs using Downward Lab 8.2 [36], with runtime/memory limits of 30min/3.5GiB. We use the benchmark set collected by Speck et al. [42], which contains 1269 problem instances with axioms. As a backend solver for SAT formulas, we use Kissat[10]. AxSAT's code[11] and the empirical results are publicly available [5].

We run AxSAT in two different search configurations, an **iterative mode**, working on one formula at a time, and a **parallel mode**, which handles multiple formulas in parallel. The iterative mode constructs formulas for increasing

[9] In our experiments (Section 6), this value is set to $5 \cdot 10^6$.
[10] https://github.com/arminbiere/kissat
[11] https://github.com/galvusdamor/decoupling-transformer-sat

		Coverage							Action Parallelism	
				Iterative		Parallel			Iterative	Parallel
Domain	#	blind	SymK	Seq	∃	Seq	∃	LAMA	∃	∃
airport-adl	50	19	19	11	21	28	**49**	40	2.24	1.98
appn-adl	33	11	23	16	**33**	**33**	**33**	**33**	8.41	8.08
assembly	30	0	11	2	16	17	**30**	**30**	4.7	4.34
blocker	7	**7**	6	**7**	6	**7**	6	4	1	0.87
blocks-axioms	35	18	30	**35**	**35**	**35**	**35**	**35**	1	0.94
cats-horndl	20	**20**	**20**	11	**20**	**20**	**20**	**20**	13.1	2.62
drones-horndl	24	**23**	16	**23**	**23**	**23**	**23**	**23**	2.17	2.25
fridge	30	0	1	0	**30**	21	**30**	**30**	26.6	26.6
ged1-ds2nd	12	**12**	8	**12**	**12**	**12**	**12**	**12**	1	0.5
ghosh-cc2	27	9	0	14	18	18	**20**	**20**	1.63	1.45
grid-axioms	5	1	3	4	3	4	4	**5**	1.33	1.08
miconic-axioms	150	60	**150**	35	**150**	**150**	**150**	**150**	15.5	6.2
miconic-fulladl	150	78	111	59	72	130	**139**	**139**	1.88	1.8
openstacks	30	7	16	5	7	20	22	**30**	1.66	1.35
openstacks-opt08-adl	30	8	**30**	3	6	**30**	**30**	**30**	1.38	1.25
openstacks-sat08-adl	25	4	12	2	3	12	14	**25**	1.37	1.33
optical-telegraphs	48	2	3	1	**48**	6	**48**	4	32.45	25.5
philosophers	48	5	12	3	**48**	12	**48**	46	25.5	22.95
psr-large	50	14	25	22	25	25	28	**44**	5.84	3.27
psr-middle	36	26	**36**	35	**36**	**36**	**36**	**36**	6.43	3.88
queens-horndl	30	23	10	**30**	**30**	**30**	**30**	**30**	3.93	2.07
robot-horndl	56	45	42	37	37	53	52	**56**	1	0.94
robotConj-horndl	56	45	43	41	41	**56**	**56**	**56**	1	0.93
snowman-basic	41	**30**	29	13	12	16	17	22	1.06	0.97
snowman-cheating	41	33	32	33	**36**	**36**	**36**	23	2.28	2.19
snowman-reach	41	**27**	25	19	17	20	18	23	1.07	0.9
sokoban-axioms	30	24	25	12	12	13	13	**28**	1	0.81
taskassign-horndl	20	**20**	12	**20**	**20**	**20**	**20**	**20**	11.95	5.51
tpsa-horndl	15	4	6	10	9	12	12	**15**	1.17	1
trucks	30	8	9	5	19	13	**25**	15	2.7	2.44
vta-horndl	15	**15**	4	13	13	**15**	**15**	**15**	1	1
vta-roles-horndl	15	**15**	0	13	13	**15**	**15**	**15**	1	1
others	39	37	37	37	37	37	37	37	2.15	2.01
$\sum$	1269	650	806	583	908	975	**1123**	1111	5.06	3.92

Table 1. Coverage results (number of instances solved) and action parallelism on the axiom benchmark set. See text for a detailed discussion.

plan-length bounds, incrementing it by 1 in each iteration, and runs the SAT solver until it terminates, i.e., shows the formula (un)satisfiable, or runs out of time/memory. Rintanen [32] proposed a parallel mode, in which solving multiple bounds is interleaved in a round-robin fashion on a single core. We use a simpler variant that in our experiments performed better: we run the formulas for different bounds sequentially, but limit each to 5min. If Kissat runs into our time or memory limit, we proceed to the next bound. This "parallel mode" increases the plan-length exponentially, i.e., iteration i runs with limit $5 \cdot (\sqrt{2})^i$, like Madagascar [30]. On the encoding side, we distinguish between the **sequential** (Seq), with one action per time step, and the **∃-step encoding** (∃).

As baselines, we compare our approach against FD's **blind** search, the symbolic search of **SymK** [41], and the first iteration of **LAMA** [29]. Note that we cannot compare ourselves against other SAT-based planner as these do not support axioms in the problem descriptions. There do exist methods based on answer-set programming and integer linear programming that support axioms,

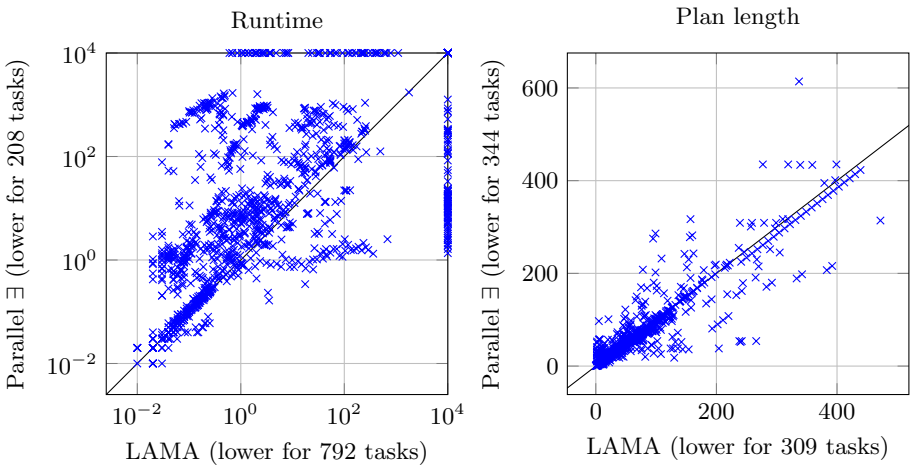

Fig. 1. Per-instance comparison of runtime (left) and plan length (right) for LAMA and our parallel ∃-step planner on the axiom benchmarks.

implemented in ASPlan and IPlan [26], respectively Plasp [9]. Unfortunately, neither ASPlan nor IPlan are publicly available. The axiom support for Plasp is very limited, only allowing for axioms that result from compiling away complex action preconditions and effects, but derived predicates are not supported. This means that Plasp is applicable only to a small fraction of our benchmarks, so we omit it in our evaluation.

The left of Table 1 shows coverage results (number of solved instances). Overall, the iterative sequential algorithm is not competitive, but it outperforms blind search and SymK significantly on a few domains (blocks, gosh, queens, tpsa). Our iterative ∃ variant outperforms both of these baselines overall, where the approaches have their strengths in different domains. Our parallel search variants improve by a lot over the iterative ones. With ∃-step encoding, our planner is overall better than LAMA, which is the state-of-the-art planner in satisficing classical planning with axioms. Taking a closer look at the two best-performing configurations, we observe that both solve all instances across many domains. On the remaining domains, they clearly have complementary strengths. Our SAT-based planner does very well in airport, optical-telegraphs, snowman-cheating, and trucks, whereas LAMA shines in openstacks, psr, snowman basic and reach, as well as sokoban. The scatter plot in Figure 1 (left) sheds further light at this comparison. It clearly confirms the complementary strengths of the two approaches, with both techniques obtaining speed-ups of several orders of magnitude over the other in different domains. Overall, our SAT-based approach shows state-of-the-art performance on domains with axioms.

In the right of Table 1 we show the action parallelism in the plans found by iterative ∃ and parallel ∃ on instances commonly solved by these two configurations. We compute the parallelism as the ratio between plan length and the

number of steps encoded in the SAT formula. Hence, the sequential encoding always obtains a parallelism of at most 1.0 (there can be steps in which no action is applied). We have two main observations: (1) in domains with high parallelism, $\exists$ significantly improves of Seq, and (2) in the same domains parallel $\exists$ is very competitive with LAMA. Another observation is that parallel $\exists$ typically achieves lower parallelism than the iterative variant, which is due to the greedy behavior of increasing the bound without waiting for the SAT solver to prove the respective formula to be unsatisfiable.

We compare the solution quality of LAMA and the parallel $\exists$ configuration in Figure 1 (right). There is no clear advantage for either of the methods. Overall, the parallel $\exists$ configuration finds shorter solutions in 344 instances (mostly in appn-adl, assembly, and robotConj-horndl) than LAMA, and longer solutions in 309 instances (mostly in airport-adl, blocks-axioms, philosophers).

We also analyzed the structure of the TDG in the benchmark instances. A substantial amount of SCCs of the TDGs has size one, which are by definition implicational as they lack internal axioms. In most domains, all derived variables are in size-1 SCCs. Exceptions are psr (only 44%, respectively 51% of SCCs are size-1), snowman-reach (46%), social-planning (86%) and sokoban (85%). These are, with the exception of social-planning, the domains in which our approach falls behind LAMA.

7 Conclusion

We introduce the first encoding for axioms and derived predicates for SAT-based planning. This closes a significant gap, as other common planning paradigms, such as heuristic explicit-state search or symbolic search, have previously supported planning with axioms. We show a sequential encoding as well as an exists-step encoding tailored to axioms, which is the standard in modern SAT-based planners. Our planner AxSAT, based on this encoding, is competitive with the state of the art in satisficing planning with axioms and performs favorably overall compared to the well-known LAMA planner.

For future work, we plan to investigate encodings specifically designed for more specialized forms of stratified axiom programs. A particularly interesting direction is to study the recent decoupled task transformation [40], which embodies decoupled search [14] and relies heavily on axioms. This includes examining the relationship between decoupled search and action parallelism ($\exists$-step encoding). Further, we will consider encodings of parallelism in planning that admit more parallelism, notably the relaxed $\exists$-step [45] and relaxed-relaxed $\exists$-step encodings [1]. Both are challenging to incorporate as they require reasoning about when and importantly how the truth values of derived predicates changes during the execution of actions *inside* of the SAT formula – in contrast to our encoding which is based on a cautious precomputed estimate.

Acknowledgments. This publication is part of the project "Exploiting Problem Structures in SAT-based Planning" of the research programme Open Competition

which is financed under the grant OCENW.M.22.050 by the Dutch Research Council (NWO) . David Speck was funded by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA). This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

1. Balyo, T.: Relaxing the relaxed exist-step parallel planning semantics. In: 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013). pp. 865–871. IEEE Computer Society (2013)
2. Behnke, G.: Block compression and invariant pruning for SAT-based totally-ordered HTN planning. In: ICAPS 2021. pp. 25–35. AAAI Press (2021). https://doi.org/10.1609/icaps.v31i1.15943
3. Behnke, G., Höller, D., Biundo, S.: totSAT – Totally-ordered hierarchical planning through SAT. In: AAAI 2018. pp. 6110–6118. AAAI Press (2018)
4. Behnke, G., Höller, D., Biundo, S.: Bringing order to chaos – A compact representation of partial order in SAT-based HTN planning. In: AAAI 2019. pp. 7520–7529. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33017520
5. Behnke, G., Speck, D., Gnad, D.: AxSAT – bringing axioms to sat planning. https://doi.org/10.5281/zenodo.15848556 (2025)
6. Bofill, M., Borralleras, C., Espasa, J., Martín, G., Patow, G., Villaret, M.: A good snowman is hard to plan. arXiv:2310.01471 [cs.AI] (2023)
7. Bofill, M., Espasa, J., Villaret, M.: The RANTANPLAN planner: system description. The Knowledge Engineering Review **31**(5), 452–464 (2016)
8. Bryce, D., Gao, S., Musliner, D.J., Goldman, R.P.: SMT-based nonlinear PDDL+ planning. In: Proc. AAAI 2015. pp. 3247–3253 (2015)
9. Dimopoulos, Y., Gebser, M., Lühne, P., Romero, J., Schaub, T.: plasp 3: Towards effective ASP planning. Theory Pract. Log. Program. **19**(3), 477–504 (2019)
10. Dimopoulos, Y., Nebel, B., Koehler, J.: Encoding planning problems in nonmonotonic logic programs. In: Proc. ECAI 1997. pp. 169–181 (1997)
11. Frisch, A.M., Giannaros, P.A.: Sat encodings of the at-most-k constraint: some old, some new, some fast, some slow (2010), manuscript
12. Gebser, M., Janhunen, T., Rintanen, J.: ASP encodings of acyclicity properties. In: Proc. KR 2014. pp. 634–637 (2014)
13. Gebser, M., Janhunen, T., Rintanen, J.: Declarative encodings of acyclicity properties. Journal of Logic and Computation **30**(4), 923–952 (2020)
14. Gnad, D., Hoffmann, J.: Star-topology decoupled state space search. AIJ **257**, 24–60 (2018)
15. Grundke, C., Röger, G., Helmert, M.: Formal representations of classical planning domains. In: Proc. ICAPS 2024. pp. 239–248 (2024)
16. Helmert, M.: The Fast Downward planning system. JAIR **26**, 191–246 (2006)
17. Helmert, M.: Concise finite-domain representations for PDDL planning tasks. AIJ **173**, 503–535 (2009)

18. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. JAIR **14**, 253–302 (2001)
19. Huang, R., Chen, Y., Zhang, W.: SAS^+ planning as satisfiability. JAIR **43**, 293–328 (2012)
20. Höller, D., Behnke, G.: Encoding lifted classical planning in propositional logic. In: ICAPS 2022. pp. 137–144. AAAI Press (2022). https://doi.org/10.1609/icaps.v32i1.19794
21. Ivankovic, F., Haslum, P.: Optimal planning with axioms. In: Proc. IJCAI 2015. pp. 1580–1586 (2015)
22. Karger, D.R., Motwani, R., Ramkumar, G.D.S.: On approximating the longest path in a graph. Algorithmica **18**(1), 82–98 (1997)
23. Kautz, H., Selman, B.: Planning as satisfiability. In: Proc. ECAI 1992. pp. 359–363 (1992)
24. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic, and stochastic search. In: Proc. AAAI 1996. pp. 1194–1201 (1996)
25. McDermott, D.: The 1998 AI Planning Systems competition. AI Magazine **21**(2), 35–55 (2000)
26. Miura, S., Fukunaga, A.: Automatic extraction of axioms for planning. In: Proc. ICAPS 2017. pp. 218–227 (2017)
27. Quenard, G., Pellier, D., Fiorino, H.: Sibylsat: Using sat as an oracle to perform a greedy search on tohtn planning. In: ECAI 2024, pp. 4157–4164. IOS Press (2024)
28. Rankooh, M.F., Rintanen, J.: Propositional encodings of acyclicity and reachability by using vertex elimination. In: Proc. AAAI 2022. pp. 5861–5868 (2022)
29. Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. JAIR **39**, 127–177 (2010)
30. Rintanen, J.: Madagascar: Scalable planning with SAT. In: IPC-8 Planner Abstracts. pp. 66–70 (2014)
31. Rintanen, J.: Temporal planning with clock-based SMT encodings. In: Proc. IJCAI 2017. pp. 743–749 (2017)
32. Rintanen, J., Heljanko, K., Niemelä, I.: Planning as satisfiability: parallel plans and algorithms for plan search. Artificial Intelligence **170**(12-13), 1031–1080 (2006)
33. Röger, G., Grundke, C.: Negated occurrences of predicates in pddl axiom bodies. In: Proceedings of the KI-2024 Workshop on Planning, Scheduling, Design and Configuration (PuK 2024) (2024)
34. Scala, E., Ramirez, M., Haslum, P., Thiébaux, S.: Numeric planning with disjunctive global constraints via smt. In: ICAPS 2016. pp. 276–284 (2016)
35. Schreiber, D.: Lilotane: A lifted SAT-based approach to hierarchical planning. JAIR **70**, 1117–1181 (2021)
36. Seipp, J., Pommerening, F., Sievers, S., Helmert, M.: Downward Lab. https://doi.org/10.5281/zenodo.790461 (2017)
37. Shaik, I., van de Pol, J.: Classical planning as QBF without grounding. In: Proc. ICAPS 2022. pp. 329–337 (2022)
38. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: Proc. CP 2005. pp. 827–831 (2005)
39. Speck, D., Geißer, F., Mattmüller, R., Torralba, Á.: Symbolic planning with axioms. In: Proc. ICAPS 2019. pp. 464–472 (2019)
40. Speck, D., Gnad, D.: Decoupled search for the masses: A novel task transformation for classical planning. In: Proc. ICAPS 2024. pp. 546–554 (2024)
41. Speck, D., Mattmüller, R., Nebel, B.: Symbolic top-k planning. In: Proc. AAAI 2020. pp. 9967–9974 (2020)

42. Speck, D., Seipp, J., Torralba, Á.: Symbolic search for cost-optimal planning with expressive model extensions. JAIR **82**, 1349–1405 (2025)
43. Thiébaux, S., Hoffmann, J., Nebel, B.: In defense of PDDL axioms. AIJ **168**(1–2), 38–69 (2005)
44. Tseitin, G.: On the complexity of derivation in the propositional calculus. In: Studies in Constructive Mathematics and Mathematical Logic, Part II, pp. 115–125. Consultants Bureau, New York (1968), english Translation
45. Wehrle, M., Rintanen, J.: Planning as satisfiability with relaxed ∃-step plans. In: Proc. AJCAI 2007. pp. 244–253 (2007)