# On the Power of Top-Down Branching Heuristics

**Matti Järvisalo**[*] and **Tommi Junttila**[†]
Helsinki University of Technology (TKK)
Department of Information and Computer Science
PO Box 5400, FI-02015 TKK, Finland
matti.jarvisalo@tkk.fi, tommi.junttila@tkk.fi

## Abstract

We study the relative best-case performance of DPLL-based structure-aware SAT solvers in terms of the power of the underlying proof systems. The systems result from (i) varying the style of branching and (ii) enforcing dynamic restrictions on the decision heuristics. Considering DPLL both with and without clause learning, we present a relative efficiency hierarchy for refinements of DPLL resulting from combinations of decision heuristics (top-down restricted, justification restricted, and unrestricted heuristics) and branching styles (typical DPLL-style and ATPG-style branching). An an example, for DPLL without clause learning, we establish a strict hierarchy, with the ATPG-style, justification restricted branching variant as the weakest system.

## Introduction

Modern complete satisfiability (SAT) solvers provide an efficient way of solving various real-world problems as propositional satisfiability. Typical SAT solvers aimed at solving such structured problems are based on the conjunctive normal form (CNF) level *Davis-Putnam-Logemann-Loveland* procedure (DPLL) (Davis and Putnam 1960; Davis, Logemann, and Loveland 1962), and often incorporate clause learning (Marques-Silva and Sakallah 1999; Beame, Kautz, and Sabharwal 2004) for boosting the efficiency of search.

A problem with CNF, however, is that as problems are translated into this low-level format, structure of the modelled problem domain is lost, and thus the SAT solver cannot make use of this structural knowledge. Indeed, in SAT based approaches, direct CNF encodings of a problem domain are rarely used, but rather, more natural representations for arbitrary propositional formulas are used during modelling. *Boolean circuits*, see e.g. (Papadimitriou 1995), provide a natural, structure-preserving representation form for modelling many typical SAT problems—e.g., bounded model checking of hardware, EDA applications like automated test pattern generation (ATPG), and automated planning. Motivated by this, there is a wide body of work on lifting the DPLL procedure to work directly on circuits, see (Junttila and Niemelä 2000; Kuehlmann, Ganai, and Paruthi 2001; Ganai et al. 2002; Thiffault, Bacchus, and Walsh 2004) for instance. A way for circuit-level solvers to exploit the structural knowledge is to use it for guiding the branching rule. One applied heuristic idea is to apply branching in a *top-down* fashion, starting from the constraints imposed on the output gates of the circuit, and to search for *justification* for the currently imposed values (Kuehlmann et al. 2002; Lu et al. 2003). A modification to the actual *style of branching* in DPLL-based algorithms, aiming at eagerly justifying the currently unjustified gates, has also been considered (Kuehlmann, Ganai, and Paruthi 2001).

This work studies the relative best-case performance of such variations of DPLL-based structure-aware Boolean circuit level SAT and ATPG solvers in terms of *proof complexity* (Beame and Pitassi 1998). In more detail, we study these solvers through the relative power of their underlying inference systems (or *proof systems*) in terms of the shortest existing proofs in the systems. For two proof systems, $S$ and $S'$, we say that $S'$ *(polynomially) simulates* $S$ if, for all infinite families $\{F_n\}$ of unsatisfiable CNF formulas, there is a polynomial that bounds for all $F_n$ the length of the shortest proofs in $S'$ w.r.t. the length of the shortest proofs in $S$. If $S'$ simulates $S$ and vice versa, then $S$ and $S'$ are *(polynomially) equivalent*. If $S'$ cannot simulate $S$ and vice versa, then $S$ and $S'$ are *incomparable*. From the practical point of view, if $S'$ cannot simulate $S$, we know that *any* implementation of $S'$ can suffer a substantial decrease in efficiency compared to implementations of $S$. For example, through a formal characterization CL of DPLL with clause learning, Beame, Kautz, and Sabharwal (2004) show that CL can provide superpolynomially shorter proofs than DPLL, and thus DPLL cannot simulate CL.

We present a relative efficiency hierarchy for variations of circuit level DPLL (with and without clause learning) resulting from combinations of branching heuristics and branching styles. Motivated by ideas for solver development, we study the variations (i) DPLL-style top-down restricted, (ii) DPLL-style justification restricted (Kuehlmann et al. 2002; Lu et al. 2003), and (iii) ATPG-style justification restricted (Kuehlmann, Ganai, and Paruthi 2001) branching

[*]Supported by Helsinki Graduate School in Computer Science and Engineering, Academy of Finland (project #122399), Emil Aaltonen Foundation, Jenny and Antti Wihuri Foundation, Nokia Foundation, and Finnish Foundation for Technology Promotion.

[†]Supported by the Academy of Finland (project #112016)

DPLL. For example, for DPLL without clause learning, we establish a strict hierarchy, with the ATPG-style branching, justification restricted DPLL variant being the weakest system. Perhaps the most surprising result obtained in this paper is that clause learning DPLL with justification restricted decisions heuristics cannot even simulate the top-down restricted variant *without clause learning*. Thus, although the idea of eagerly and locally justifying the values of currently unjustified constraints is an intuitively appealing one, it can lead to dramatic losses in the best-case efficiency of a structure-aware SAT solver even when the powerful search space pruning technique of clause learning is applied.

## Preliminaries

### Boolean Circuits and SAT

A Boolean circuit over a finite set $G$ of *gates* is a set $\mathcal{C}$ of equations of form $g := f(g_1, \ldots, g_n)$, where $g, g_1, \ldots, g_n \in G$ and $f : \{\mathbf{f}, \mathbf{t}\}^n \to \{\mathbf{f}, \mathbf{t}\}$ is a Boolean function, with the additional requirements that (i) each $g \in G$ appears at most once as the left hand side in the equations in $\mathcal{C}$, and (ii) the underlying directed graph $\langle G, E(\mathcal{C}) = \{\langle g', g\rangle \in G \times G \mid g := f(\ldots, g', \ldots) \in \mathcal{C}\}\rangle$ is acyclic. If $\langle g', g\rangle \in E(\mathcal{C})$, then $g'$ is a *child* of $g$ and $g$ is a *parent* of $g'$. If $g := f(g_1, \ldots, g_n)$ is in $\mathcal{C}$, then $g$ is an $f$-gate (or of type $f$), otherwise it is an *input gate*. A gate with no parents is an *output gate*.

A (partial) assignment for $\mathcal{C}$ is a (partial) function $\tau : G \to \{\mathbf{f}, \mathbf{t}\}$. An assignment $\tau$ is *consistent with* $\mathcal{C}$ if $\tau(g) = f(\tau(g_1), \ldots, \tau(g_n))$ for each $g := f(g_1, \ldots, g_n)$ in $\mathcal{C}$. Under a (possibly partial) assignment $\tau$, (i) a gate $g$ is *assigned* if $\tau(g)$ is defined, and (ii) an assigned gate is *justified* if it is an input gate or $g := f(g_1, \ldots, g_n)$ and $\forall \tau' \supseteq \tau : \tau(g) = f(\tau'(g_1), \ldots, \tau'(g_n))$ holds. That is, the current values of the children of a justified gate are enough for the gate to evaluate to its value.

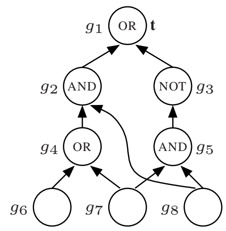
A *constrained Boolean circuit* $\langle \mathcal{C}, \tau\rangle$ is a pair $\langle \mathcal{C}, \tau\rangle$, where $\mathcal{C}$ is a Boolean circuit and $\tau$ is a partial assignment for $\mathcal{C}$. With respect to a $\langle \mathcal{C}, \tau\rangle$, each $\langle g, v\rangle \in \tau$ is a *constraint*, and $g$ is *constrained* to $v$ if $\langle g, v\rangle \in \tau$. An assignment $\tau'$ *satisfies* $\langle \mathcal{C}, \tau\rangle$ if (i) $\tau'$ is consistent with $\mathcal{C}$, and (ii) $\tau' \supseteq \tau$. If some assignment satisfies $\langle \mathcal{C}, \tau\rangle$ then $\langle \mathcal{C}, \tau\rangle$ is *satisfiable* and otherwise *unsatisfiable*.

For convenience, we restrict the set of Boolean functions that can be used as gate types to the following.

- $\textsc{not}(v)$ is $\mathbf{t}$ iff $v$ is $\mathbf{f}$.
- $\textsc{or}(v_1, \ldots, v_n)$ is $\mathbf{t}$ iff at least one of $v_1, \ldots, v_n$ is $\mathbf{t}$.
- $\textsc{and}(v_1, \ldots, v_n)$ is $\mathbf{t}$ iff all $v_1, \ldots, v_n$ are $\mathbf{t}$.

**Example 1** *A constrained Boolean circuit is shown in Fig. 1. One satisfying assignment for it is* $\tau' = \{\langle g_1, \mathbf{t}\rangle, \langle g_2, \mathbf{t}\rangle, \langle g_3, \mathbf{f}\rangle, \langle g_4, \mathbf{t}\rangle, \langle g_5, \mathbf{t}\rangle, \langle g_6, \mathbf{f}\rangle, \langle g_7, \mathbf{t}\rangle, \langle g_8, \mathbf{t}\rangle\}$. *Under the partial assignment* $\{\langle g_1, \mathbf{t}\rangle, \langle g_2, \mathbf{t}\rangle, \langle g_4, \mathbf{t}\rangle, \langle g_8, \mathbf{t}\rangle\}$*, the gates* $g_1$*,* $g_2$*, and* $g_8$ *are justified while the gate* $g_4$ *is assigned but unjustified.*

We apply the standard "Tseitin translation" to map each constrained Boolean circuit $\langle \mathcal{C}, \tau\rangle$ into an equi-satisfiable CNF formula $\mathsf{cnf}(\langle \mathcal{C}, \tau\rangle)$. First, introduce a variable $\tilde{g}$ for




$$\begin{aligned}\mathcal{C} \;=\; \{&g_1 := \textsc{or}(g_2, g_3)\\ &g_2 := \textsc{and}(g_4, g_8)\\ &g_3 := \textsc{not}(g_5)\\ &g_4 := \textsc{or}(g_6, g_7)\\ &g_5 := \textsc{and}(g_7, g_8)\}\end{aligned}$$

$$\tau = \{\langle g_1, \mathbf{t}\rangle\}$$

Figure 1: A constrained Boolean circuit $\langle \mathcal{C}, \tau\rangle$.

each gate $g$ in the circuit. Then, describe the functionality of each gate and the constraints with clauses (Table 1). When convenient, we view a clause as a finite set of literals and a CNF formula as a finite set of clauses.

Table 1: The CNF translation $\mathsf{cnf}(\langle \mathcal{C}, \tau\rangle)$

| gate or constraint in $\langle \mathcal{C}, \tau\rangle$ | clauses in $\mathsf{cnf}(\langle \mathcal{C}, \tau\rangle)$ |
|---|---|
| $g := \textsc{not}(g_1)$ | $\{\neg\tilde{g}, \neg\tilde{g}_1\}, \{\tilde{g}, \tilde{g}_1\}$ |
| $g := \textsc{or}(g_1, \ldots, g_n)$ | $\{\neg\tilde{g}, \tilde{g}_1, \ldots, \tilde{g}_n\}, \{\tilde{g}, \neg\tilde{g}_1\}, \ldots, \{\tilde{g}, \neg\tilde{g}_n\}$ |
| $g := \textsc{and}(g_1, \ldots, g_n)$ | $\{\neg\tilde{g}, \tilde{g}_1\}, \ldots, \{\neg\tilde{g}, \tilde{g}_n\}, \{\tilde{g}, \neg\tilde{g}_1, \ldots, \neg\tilde{g}_n\}$ |
| $\langle g, \mathbf{t}\rangle \in \tau$ | $\{\tilde{g}\}$ |
| $\langle g, \mathbf{f}\rangle \in \tau$ | $\{\neg\tilde{g}\}$ |

Any CNF formula $F = \{C_1, \ldots, C_k\}$ can be seen as a *CNF circuit* $\mathsf{circ}(F)$. Take an input gate $g_x$ for each variable $x$ in $F$. Now $\mathsf{circ}(F)$ is $\{g_F := \textsc{and}(g_{C_1}, \ldots, g_{C_k})\} \cup \{g_{C_i} := \textsc{or}(g_{l_1}, \ldots, g_{l_m}) \mid C_i = \{l_1, \ldots, l_m\} \in F\} \cup \{g_{\neg x} := \textsc{not}(g_x) \mid \neg x \in \cup_{i=1}^{k} C_i\}$. The *constrained CNF circuit* $\mathsf{ccirc}(F) := \langle \mathsf{circ}(F), \{\langle g_F, \mathbf{t}\rangle\}\rangle$ is satisfiable iff $F$ is.

### Resolution

The well-known Resolution proof system (RES) is based on the *resolution rule*. Let $C, D$ be clauses, and $x$ a Boolean variable. The resolution rule lets us derive the clause $C \cup D$ from the clauses $\{x\} \cup C$ and $\{\neg x\} \cup D$ by *resolving on* $x$. A RES *proof (for the unsatisfiability) of a CNF formula* $F$ is a sequence of clauses $\pi = (C_1, C_2, \ldots, C_m = \emptyset)$, where each $C_i$, $1 \le i \le m$, is either (i) a clause in $F$ (an *initial clause*), or (ii) derived with the resolution rule from two clauses $C_j, C_k$ where $1 \le j, k < i$ (a *derived clause*). The *length* of $\pi$ is $m$, the number of clauses occurring in it.

Many *refinements of* Resolution, in which the structure of RES proofs is restricted, have been studied. Here of particular interest is *Tree-like Resolution* (T-RES) that requires the refutations to be representable as trees.

Superpolynomial lower bounds on proof length in RES have been shown for various families of CNF formulas. One such family is the *pigeon-hole principle*: $m$ pigeons cannot sit in $n$ holes so that every pigeon has its own hole if $n < m$. We consider the case $m = n+1$ encoded as the CNF formula

$$\mathrm{PHP}_n^{n+1} := \bigwedge_{i=1}^{n+1} \Big(\bigvee_{j=1}^{n} p_{i,j}\Big) \wedge \bigwedge_{j=1}^{n} \bigwedge_{i=1}^{n} \bigwedge_{i'=i+1}^{n+1} (\neg p_{i,j} \vee \neg p_{i',j}),$$

where each $p_{i,j}$ is a Boolean variable with the interpretation "$p_{i,j}$ is $\mathbf{t}$ if and only if the $i^{\text{th}}$ pigeon sits in the $j^{\text{th}}$ hole".

**Theorem 1 (Haken (1985))** *There are no polynomial length* RES *proofs for the family* $\{\mathrm{PHP}_n^{n+1}\}$.

It is also known that T-RES is a *proper* refinement of RES.

**Corollary 1 (Ben-Sasson, Impagliazzo, and Wigderson)** T-RES *cannot polynomially simulate* RES.

## DPLL and Clause Learning

Most modern complete SAT solvers are based on DPLL (Davis and Putnam 1960; Davis, Logemann, and Loveland 1962). Given a CNF formula $F$, DPLL is a depth-first search procedure building a partial assignment for the variables in $F$ through (i) *branching* and (ii) *unit propagation* (UP). In branching, the current assignment is extended with the assignment (*decision*) $\langle x, v\rangle$, where $v \in \{\mathbf{f}, \mathbf{t}\}$, for some unassigned variable $x$. Unit propagation refers to applying the *unit clause rule*: if there is a clause $(l_1 \vee \cdots \vee l_k \vee l) \in F$ and assignments $\langle l_i, \mathbf{f}\rangle$ for each $1 \leq i \leq k$, the current partial assignment can be extended with $\langle l, \mathbf{t}\rangle$.

An assignment is extended until (i) some variable $x$ would be assigned both **f** and **t** (a *conflict* is reached, with $x$ as the *conflict variable*) or (ii) the current assignment satisfies $F$ (in which case DPLL terminates). In case (i), non-clause learning DPLL solvers *backtrack* to the last branching decision which has not been backtracked upon, undoing all assignments made by UP after the particular decision, and flip the decision. DPLL terminates on an unsatisfiable CNF formula when there are no untried branches left.

It is well-known that DPLL and T-RES can polynomially simulate each other.

**Fact 1** DPLL *and* T-RES *are polynomially equivalent.*

Clause learning DPLL algorithms differ from non-clause learning algorithms in what happens when reaching a conflict. If a conflict is reached without any branching, the formula $F$ is determined unsatisfiable. Otherwise, the conflict is *analyzed* based on a *conflict graph*, and a *learned clause* (or *conflict clause*), which describes the "cause" of the conflict, is added to $F$. After this the search is continued typically by applying *non-chronological backtracking* (or *conflict-driven backjumping*) for backtracking to an earlier decision level that "caused" the conflict. Conflict-driven backjumping results in the fact that, as opposed to the basic backtracking in DPLL, the other branch (opposite value) of decision variables is not necessary forced systematically when backtracking. In other words, branching in CL is seen simply as assigning values to unassigned variables, rather than as a branching rule in which by branching on a variable $x$ the current branch is always extended into two branches, one with $x$ and the other with $\neg x$.

For investigating the efficiency of clause learning DPLL in proof complexity theoretic terms, we apply the characterization of Beame, Kautz, and Sabharwal (2004), referred to as the CL *proof system*. A clause learning proof (or CL proof) induced by a learning scheme $S$ is constructed by applying branching, applying unit propagation whenever possible, and using $S$ to learn conflict clauses when conflicts are reached, so that in the end, a conflict can be reached at decision level zero. While the efficiency gains obtained in practice by implementing clause learning in DPLL based algorithms are well-established, (Beame, Kautz, and Sabharwal 2004) provides the first formal study on its power: CL cannot be simulated by any refinement of RES that cannot itself simulate RES.

**Corollary 2 (Beame, Kautz, and Sabharwal (2004))** DPLL *cannot simulate* CL.

On the other hand, even with unlimited restarts, CL is at most as powerful as RES.

**Theorem 2 (Beame, Kautz, and Sabharwal (2004))** RES *can simulate* CL *even if* CL *is allowed unlimited restarts.*

Notice that CL does not include restarts as such. In the following, we explicitly mention when results hold even when restarts are allowed.

**Circuit Level DPLL and CL.** From the viewpoint of DPLL based search, there is a tight correspondence between a constrained Boolean circuit $\langle \mathcal{C}, \tau\rangle$ and its CNF translation $\mathsf{cnf}(\langle \mathcal{C}, \tau\rangle)$ in Table 1. The CNF translation has a one-to-one correspondence between the gates and the CNF variables, and encodes in a natural way the semantics of the gates; thus circuit level Boolean constraint propagation (see (Junttila and Niemelä 2000; Kuehlmann, Ganai, and Paruthi 2001; Ganai et al. 2002; Thiffault, Bacchus, and Walsh 2004)) on $\langle \mathcal{C}, \tau\rangle$ corresponds to unit clause propagation on $\mathsf{cnf}(\langle \mathcal{C}, \tau\rangle)$. For example, consider the gate $g := \textsc{and}(g_1, g_2)$ and its CNF translation $(\neg\tilde{g} \vee \tilde{g}_1) \wedge (\neg\tilde{g} \vee \tilde{g}_2) \wedge (\tilde{g} \vee \neg\tilde{g}_1 \vee \neg\tilde{g}_2)$. Now whenever the gate $g_2$ is assigned to **f**, the gate $g$ can be propagated to **f** by the semantics of AND. On the CNF level, we can equivalently propagate the variable $\tilde{g}$ to **f** by applying the unit clause rule whenever the variable $\tilde{g}_2$ is assigned to **f**. Due to this correspondence, clause learning can also be equivalently applied in circuit level SAT solvers for learning conflict clauses. Therefore, here we consider proof systems like DPLL and CL to work on circuit level and write, e.g., "a CL proof of $\langle \mathcal{C}, \tau\rangle$" instead of "a CL proof of $\mathsf{cnf}(\langle \mathcal{C}, \tau\rangle)$".

# Top-Down Branching DPLL

One often applied heuristic idea is to branch on variables *top-down* with respect to the circuit structure, starting from the constraints imposed on the output gates of the circuit, and searching for *justification* for the currently assigned values. We characterize the variants of this idea through two *dynamic branching restrictions*:

**Top-down restriction:** Branching is allowed on gate $g$ if $g$ has a currently assigned parent. These variants of DPLL and CL are denoted by $\mathsf{DPLL}_{\mathsf{td}}$ and $\mathsf{CL}_{\mathsf{td}}$.

**Justification-based restriction:** Branching is allowed on gate $g$ if $g$ has a currently assigned and unjustified parent. These variants of DPLL and CL are $\mathsf{DPLL}_{\mathsf{jf}}$ and $\mathsf{CL}_{\mathsf{jf}}$.

A modification to the actual *style* of branching in DPLL-based algorithm, heuristically aiming at justifying the current unjustified assignments on gates, has been considered especially in Boolean circuit level SAT solvers for ATPG. The underlying $\mathsf{DPLL}_{\mathsf{jf}}^{\mathsf{atpg}}$ system using ATPG-style branching is a variation of the justification-based restricted branching $\mathsf{DPLL}_{\mathsf{jf}}$. The difference between original DPLL-style branching and ATPG-style branching is illustrated in Fig. 2 with an OR-gate $g := \textsc{or}(g_1, g_2, g_3)$. Where original DPLL-style branching is based on branching on a variable

(Fig. 2 left), in ATPG-style branching (Fig. 2 right) each branch will have a unique justification for the currently assigned value of the parent ($g$ is **t** in the example).

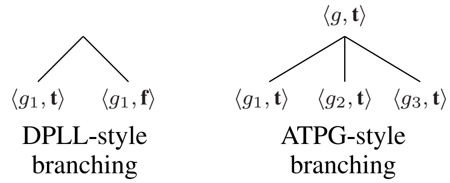



Figure 2: Styles of branching; OR-gate $g := \textsc{or}(g_1, g_2, g_3)$

## Proof Complexity

In this section we present the main results of this work. First we study the relative efficiency of $\mathsf{DPLL}_{\mathsf{td}}$, $\mathsf{DPLL}_{\mathsf{jf}}$, and $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ w.r.t. $\mathsf{DPLL}$. After this, we turn to the case of clause learning. The results are summarized in Fig. 3. In the hierarchy, a system $S$ cannot simulate $S'$ if there is an arrow from $S$ to $S'$ with a line crossed over. A plain arrow from $S$ to $S'$ means that $S$ can simulate $S'$. Arrows labelled with $\star$ are known results from (Beame, Kautz, and Sabharwal 2004; Järvisalo, Junttila, and Niemelä 2005); the unlabeled ones are results of this paper. The arrows induced by the transitivity of negative/positive simulation results are left out for clarity.

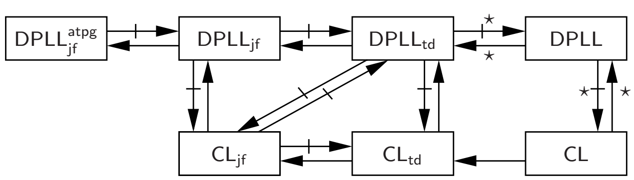



Figure 3: Summary of results

### $\mathsf{DPLL}$ vs $\mathsf{DPLL}_{\mathsf{td}}$ vs $\mathsf{DPLL}_{\mathsf{jf}}$ vs $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$

The relative efficiency of $\mathsf{DPLL}_{\mathsf{td}}$ and $\mathsf{DPLL}$ has been studied in (Järvisalo, Junttila, and Niemelä 2005): while $\mathsf{DPLL}$ trivially simulates $\mathsf{DPLL}_{\mathsf{td}}$, $\mathsf{DPLL}_{\mathsf{td}}$ cannot simulate $\mathsf{DPLL}$.

**Theorem 3 (Järvisalo, Junttila, and Niemelä (2005))** $\mathsf{DPLL}_{\mathsf{td}}$ *cannot polynomially simulate* $\mathsf{DPLL}$.

We now consider the pairwise relative efficiency of the other variations of $\mathsf{DPLL}$. First, we observe that when restricting to constrained CNF circuits, $\mathsf{DPLL}$, $\mathsf{DPLL}_{\mathsf{td}}$, and $\mathsf{DPLL}_{\mathsf{jf}}$ are equivalent.

**Lemma 1** *For constrained CNF circuits,* $\mathsf{DPLL}$, $\mathsf{DPLL}_{\mathsf{td}}$, *and* $\mathsf{DPLL}_{\mathsf{jf}}$ *are equivalent.*

*Proof sketch.* Given an arbitrary constrained CNF circuit, after unit propagation on the output gate $\mathsf{DPLL}$ and $\mathsf{DPLL}_{\mathsf{td}}$ can branch on all of the input gates. If there is an input gate on which $\mathsf{DPLL}_{\mathsf{jf}}$ cannot branch on, all of its parents have already been justified, and hence branching on such gate in any of the systems would be redundant. $\square$

To separate $\mathsf{DPLL}_{\mathsf{jf}}$ and $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$, we use a known result on the efficiency of *clausal tableaux*. A *clausal tableau* $T$ for a set of clauses $F$ is a tree in which a set of clauses is associated with each node in $T$. The original set of clauses $F$ is associated with the root of $T$. Each internal node $v$ in $T$, with the associated set of clauses $F_v$, has exactly $k$ children, and the set of clauses associated with the $i$th child $v_i$ is $F_v \cup \{(l_i)\}$, where a $C_j = (l^j_1 \vee \ldots \vee l^j_k) \in F_v$ defines $k$ and the literals $l^j_1, \ldots, l^j_k$ ($C_k$ is *decomposed*). A branch (path) in the tableau is closed if some variable occurs both positively and negatively in the set of unit clauses associated with the leaf node of the branch. Any clausal tableau for a set of clauses $F$ in which all branches in the tableau are closed, is a clausal tableau proof (for the unsatisfiability) of $F$. The proof system $\mathsf{CT}$ consists of all clausal tableau proofs.

It is known that $\mathsf{CT}$ is not as powerful as $\mathsf{T\text{-}RES}$.

**Theorem 4 (Arai, Pitassi, and Urquhart (2001))** $\mathsf{CT}$ *cannot simulate* $\mathsf{T\text{-}RES}$.

Now, $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ and $\mathsf{CT}$ are equivalent in the sense that, given an arbitrary set of clauses $F$, the minimal length proofs for $\mathsf{ccirc}(F)$ in $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ are polynomially bounded in the minimal length proofs for $F$ in $\mathsf{CT}$, and vice versa.

**Lemma 2** *For sets of clauses,* $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ *and* $\mathsf{CT}$ *are equivalent.*

*Proof sketch.* Given an arbitrary set of clauses $F$, notice that after unit propagation on the output gate of $\mathsf{ccirc}(F)$, branching in $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ and extending a branch in $\mathsf{CT}$ are effectively equivalent on the clauses in $F$.

Now assume that unit propagation in $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ assigned a gate $g_l$ to **t**. There then is a clause $C = \{l_1, \ldots, l_k, l\} \in F$ such that all $\neg l_i$'s and $\langle g_{l_i}, \mathbf{f} \rangle$'s are in the branch for $\mathsf{CT}$ and $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$, respectively. To simulate unit propagation in $\mathsf{CT}$, decompose $C$ to its literals. Due to the opposite literals $\neg l_i$ in the branch, each branch with $l_i$ is now closed. $\square$

Theorem 4 and Fact 1 imply that $\mathsf{CT}$ cannot simulate $\mathsf{DPLL}$. Thus by Lemma 2, together with the fact that $\mathsf{DPLL}$ and $\mathsf{DPLL}_{\mathsf{jf}}$ are equivalent on constrained CNF circuits (Lemma 1), we have the following.

**Corollary 3** $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ *cannot simulate* $\mathsf{DPLL}_{\mathsf{jf}}$.

To the other direction, however, we have a positive results.

**Theorem 5** $\mathsf{DPLL}_{\mathsf{jf}}$ *can simulate* $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$.

*Proof sketch.* Assume that $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ branches with extensions $\langle g_1, v \rangle$, ..., $\langle g_k, v \rangle$, where we have a gate $g := f(g_1, \ldots, g_k)$ (if $f = \textsc{and}$ ($f = \textsc{or}$) then $v = \mathbf{f}$ ($v = \mathbf{t}$, respectively)). Simulate this in $\mathsf{DPLL}_{\mathsf{jf}}$ by branching with $g_1$ and consecutively on $g_i$ from $i = 2$ to $k-1$ in the branch having $\langle g_j, \neg v \rangle$ for all $j < i$. Unit propagate to get $\langle g_k, v \rangle$ in the branch having $\langle g_i, \neg v \rangle$ for all $i < k$. $\square$

We still have to consider whether $\mathsf{DPLL}_{\mathsf{jf}}$ can simulate $\mathsf{DPLL}_{\mathsf{td}}$. This turns out not to be the case. In order to construct a witness for this separation, we modify a construction from (Järvisalo and Junttila 2007). The construction is based on $\mathsf{circ}(\mathrm{PHP}^{n+1}_n)$. Cook (1976) introduces a polynomial number of clauses which, interpreted as adding gates to $\mathsf{ccirc}(\mathrm{PHP}^{n+1}_n)$, enable polynomial length proofs in $\mathsf{RES}$ for the resulting circuit. As a circuit structure, this *extension* is defined as $\mathrm{EXT}_n := \bigcup_{l=1}^{n} \mathrm{EXT}^l$, where

$$\mathrm{EXT}^l := \bigcup_{i=1}^{l} \bigcup_{j=1}^{l-1} \{e^l_{i,j} {:=} \textsc{or}(e^{l+1}_{i,j}, o^l_{i,j}), o^l_{i,j} {:=} \textsc{and}(e^{l+1}_{i,l}, e^{l+1}_{l+1,j})\},$$

and each $e^n_{i,j}$ is the input gate $g_{p_{i,j}}$ associated with the variable $p_{i,j}$ in $\mathrm{PHP}^{n+1}_n$. By (Cook 1976) we immediately have a polynomial length $\mathsf{RES}$ proof $\pi = (C_1, \ldots, C_m = \emptyset)$ for $\mathsf{cnf}(\mathsf{ccirc}(\mathrm{PHP}^{n+1}_n) \cup \mathrm{EXT}_n)$. In (Järvisalo and Junttila 2007), in order to guarantee short T-RES (and hence short DPLL) proofs for the construction, an additional structure (called $\mathrm{E}(\pi)$, see (Järvisalo and Junttila 2007) for details) is added to the circuit. We apply a slightly modified version (basically, the direction of the $h_i$ AND-gate chain is reversed) of $\mathrm{E}(\pi)$ to ensure short $\mathsf{DPLL_{td}}$ proofs as well:

$$\begin{aligned}\mathrm{P}(\pi) \quad := \quad & \cup_{i=1}^{m-2}\{h_i := \text{AND}(g_{C_i}, h_{i+1})\} \cup \\ & \cup_{i=1}^{m-1}\{g_{C_i} := \text{OR}(g_1, \ldots, g_j, \hat{g}_{j+1}, \ldots, \hat{g}_k) \mid \\ & \qquad C_i = \{\tilde{g}_1, \ldots, \tilde{g}_j, \neg\tilde{g}_{j+1}, \ldots, \neg\tilde{g}_k\}\} \cup \\ & \cup_{i=1}^{m-1}\{\hat{g} := \text{NOT}(g) \mid \neg\tilde{g} \in C_i\},\end{aligned}$$

where $h_{m-1}$ is the gate $g_{C_{m-1}}$. The structure of $\mathrm{P}(\pi)$ is illustrated in Fig. 4(left).

To get our final construct $\mathrm{PPHP}^{n+1}_n$ (see Fig.4(center) for clarity), we will add to the construct $\mathsf{circ}(\mathrm{PHP}^{n+1}_n) \cup \mathrm{EXT}_n \cup \mathrm{P}(\pi)$ the gates $y := \text{OR}(h_1, z)$ and $x := \text{AND}(y, z)$, where $z$ is the output gate of $\mathsf{circ}(\mathrm{PHP}^{n+1}_n)$, and constrain the output gate $x$ to $\mathbf{t}$. The idea behind $\mathrm{PPHP}^{n+1}_n$ is that $\mathrm{P}(\pi)$ encodes $\pi$ in a way that allows polynomial length $\mathsf{DPLL_{td}}$ proofs for $\mathrm{PPHP}^{n+1}_n$, while the additional structure on top of $\mathsf{circ}(\mathrm{PHP}^{n+1}_n) \cup \mathrm{EXT}_n \cup \mathrm{P}(\pi)$ prevents $\mathsf{DPLL_{jf}}$ from having polynomial proofs for $\mathrm{PPHP}^{n+1}_n$.

**Theorem 6** $\mathsf{DPLL_{jf}}$ *cannot simulate* $\mathsf{DPLL_{td}}$.

*Proof sketch.* $\mathsf{DPLL_{td}}$ has polynomial length proofs for $\mathrm{PPHP}^{n+1}_n$ w.r.t. $n$. After unit propagation, both $y$ and $z$ are $\mathbf{t}$. Then branch on $h_1$. The branch with $\langle h_1, \mathbf{t}\rangle$ propagates to $\langle h_i, \mathbf{t}\rangle$ for all $i < m$. At the latest, when assigning $\langle h_{m-1}, \mathbf{t}\rangle$, unit propagation gives a conflict, since $\mathrm{P}(\pi)$ encodes the two contradictory unit clauses in $\pi$. Consecutively from $i = 1$ to $m - 2$, branch on $g_{C_i}$ in the branch having $\langle h_k, \mathbf{f}\rangle$ for all $k \leq i$ and $\langle g_{C_j}, \mathbf{t}\rangle$ for all $j < i$. The branch with $\langle g_{C_j}, \mathbf{t}\rangle$ for all $j < i$ and $\langle g_{C_i}, \mathbf{f}\rangle$ propagates to a conflict, since $C_i$ has been derived from $C_k, C_l \in \pi$ with $k, l < i$ and we have $\langle g_{C_k}, \mathbf{t}\rangle$ and $\langle g_{C_l}, \mathbf{t}\rangle$ (for the base case, for each $C \in \mathrm{PHP}^{n+1}_n$ we have $\langle g_C, \mathbf{t}\rangle$). The branch with $\langle h_k, \mathbf{f}\rangle$ for all $k < m-1$ and $\langle g_{C_i}, \mathbf{t}\rangle$ for all $i < m-1$ propagates to $\langle g_{C_{m-1}}, \mathbf{f}\rangle$, and again we have a conflict as above. This concludes the polynomial length proof for $\mathsf{DPLL_{td}}$.

Now consider $\mathsf{DPLL_{jf}}$. Propagation gives $\langle y, \mathbf{t}\rangle$ and $\langle z, \mathbf{t}\rangle$ from $\langle x, \mathbf{t}\rangle$, and $\langle z, \mathbf{t}\rangle$ justifies $\langle y, \mathbf{t}\rangle$. Now we can branch on the inputs in $\mathrm{PHP}^{n+1}_n$ only. Moreover, $h_1$ is redundant in the sense that it is not constrained and thus cannot contribute to conflicts based on values propagated bottom-up from the input gates. Thus any $\mathsf{DPLL_{jf}}$ proof of $\mathrm{PPHP}^{n+1}_n$ must effectively include a proof of $\mathsf{ccirc}(\mathrm{PHP}^{n+1}_n)$. Since $\mathsf{DPLL}$ simulates $\mathsf{DPLL_{jf}}$, Theorem 1 and Fact 1 imply that $\mathsf{DPLL_{jf}}$ has no polynomial length proofs for $\mathrm{PPHP}^{n+1}_n$. □

The upper part of Fig. 3 summarizes the results this far.

## On the Relative Efficiency of $\mathsf{CL}$, $\mathsf{CL_{td}}$, and $\mathsf{CL_{jf}}$

We now turn to the case of clause learning solvers, and study the relative efficiency of $\mathsf{CL_{jf}}$ and $\mathsf{CL_{td}}$ w.r.t. $\mathsf{CL}$ and $\mathsf{DPLL}$. Before detailed results, we use the construction of (Järvisalo, Junttila, and Niemelä 2005) in the proof of Theorem 3 to explain why a separation of $\mathsf{DPLL}$ and $\mathsf{DPLL_{td}}$ does not directly imply a separation between $\mathsf{CL}$ and $\mathsf{CL_{td}}$.

**Example 2** *Define a circuit gadget*

$$\begin{aligned}\mathrm{TD}_n := & \{v{:=}\text{OR}(v_1, w_1)\} \cup \{v_i{:=}\text{AND}(x_i, z_i) \mid 1 \leq i \leq n\} \cup \\ & \{w_i{:=}\text{AND}(y_i, z_i), z_i{:=}\text{OR}(v_{i+1}, w_{i+1}) \mid 1 \leq i \leq n\}\end{aligned}$$

*and let* $\mathrm{UNSAT}^x := \{\{\neg x_1, \neg x_2\}, \{\neg x_1, x_2\}, \{x_1, \neg x_2\}, \{x_1, x_2\}\}$. *Now take the unsatisfiable constrained circuit*

$$\mathrm{TDU}_n := \langle \mathrm{TD}_n \cup \mathsf{circ}(\mathrm{UNSAT}^a) \cup \mathsf{circ}(\mathrm{UNSAT}^b), \{\langle v, \mathbf{t}\rangle\}\rangle$$

*with the output gates of* $\mathsf{circ}(\mathrm{UNSAT}^a)$ *and* $\mathsf{circ}(\mathrm{UNSAT}^b)$ *identified with* $v_{n+1}$ *and* $w_{n+1}$*, respectively, as illustrated in Fig. 4(right). Since unit propagation sets values to all the other gates once the input gates are assigned, branching on the input gates corresponding to* $a_1, a_2, b_1, b_2$ *gives a linear size* $\mathsf{DPLL}$ *proof for* $\mathrm{TDU}_n$*. One can similarly construct a small* $\mathsf{CL}$ *proof. For* $\mathsf{DPLL_{td}}$, $\mathsf{DPLL_{jf}}$, *and* $\mathsf{DPLL^{atpg}_{jf}}$*, the minimal proofs are of exponential length; the structure of* $\mathrm{TD}_n$ *forces them to branch on* $v_i$ *or* $w_i$ *for each* $i$ *from* $1$ *to* $n+1$*. This results in an exponential number of branches as a contradiction can be reached only in the* $\mathsf{circ}(\mathrm{UNSAT}^a)$ *and* $\mathsf{circ}(\mathrm{UNSAT}^b)$ *parts.*

*However,* $\mathsf{CL_{td}}$ *and* $\mathsf{CL_{jf}}$ *both have proofs of linear length w.r.t.* $n$ *for* $\mathrm{TDU}_n$*. First, consecutively from* $i = 1$ *to* $n+1$ *branch with* $\langle v_i, \mathbf{t}\rangle$*. Next branch with* $\langle a_1, \mathbf{t}\rangle$*. This propagates to a conflict, the clause* $\{\neg C_1, \neg C_2, \neg a_1\}$ *is learned (*$C_1$ *and* $C_2$ *are the* OR*-gates corresponding to the clauses* $\{\neg a_1, \neg a_2\}$ *and* $\{\neg a_1, a_2\}$ *in* $\mathrm{UNSAT}^a$*), the search backjumps to the previous decision level, and propagation on the learned clause gives* $\langle a_1, \mathbf{f}\rangle$*. This in turn produces a conflict, the unit clause* $\{\neg v_{n+1}\}$ *is learned, and the search backjumps to decision level 0. The same process is repeated for the right part of the circuit (replace* $v$ *with* $w$ *and* $a$ *with* $b$*). Finally, a conflict with the output constraint is reached at decision level 0 by propagating* $\{\neg v_{n+1}\}$ *and* $\{\neg w_{n+1}\}$ *up the circuit structure. Hence, while* $\mathrm{TDU}_n$ *separates* $\mathsf{DPLL}$ *from* $\mathsf{DPLL_{td}}$, $\mathsf{DPLL_{jf}}$, *and* $\mathsf{DPLL^{atpg}_{jf}}$*, it does not do the same for the corresponding clause learning extensions.*

**Lemma 3** $\mathsf{CL_{jf}}$ *has no polynomial length proofs for* $\{\mathrm{PPHP}^{n+1}_n\}$*. This holds even if restarts are allowed.*

*Proof sketch.* Through a similar argument as in the case of $\mathsf{DPLL_{jf}}$ in the proof of Theorem 6, any $\mathsf{CL_{jf}}$ proof of $\mathrm{PPHP}^{n+1}_n$ must effectively include a proof of $\mathsf{ccirc}(\mathrm{PHP}^{n+1}_n)$. Hence Theorems 1 and 2 now imply that $\mathsf{CL_{jf}}$ has no polynomial length proofs for $\mathrm{PPHP}^{n+1}_n$. □

**Corollary 4** $\mathsf{CL_{jf}}$ *cannot simulate* $\mathsf{DPLL_{td}}$*. This holds regardless of whether restarts are allowed.*

Through a similar argument as in Lemma 1, $\mathsf{CL}$, $\mathsf{CL_{td}}$, and $\mathsf{CL_{jf}}$ are equivalent for constrained CNF circuits.

**Lemma 4** *For constrained CNF circuits,* $\mathsf{CL}$, $\mathsf{CL_{td}}$, *and* $\mathsf{CL_{jf}}$ *are equivalent. This holds even if each system is allowed unlimited restarts.*

By Fact 1, Corollary 2, and Lemmas 1 and 4, we arrive at:

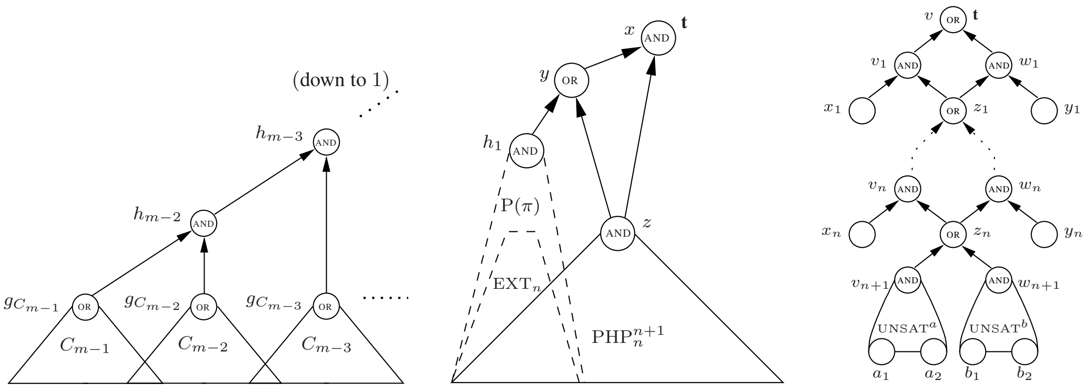



Figure 4: From left to right: high-level views for the constructs $\mathrm{P}(\pi)$, $\mathrm{PPHP}_n^{n+1}$, and $\mathrm{TDU}_n$.

**Corollary 5** $\mathsf{DPLL}_{\mathsf{td}}$ *cannot simulate* $\mathsf{CL}_{\mathsf{jf}}$.

Thus $\mathsf{CL}_{\mathsf{jf}}$ and $\mathsf{DPLL}_{\mathsf{td}}$ are polynomially incomparable.

**Theorem 7** $\mathsf{CL}_{\mathsf{jf}}$ *and* $\mathsf{DPLL}_{\mathsf{td}}$ *are incomparable.*

Finally, we end up with the relative efficiency hierarchy shown in Fig. 3. The only remaining open question in the hierarchy is whether $\mathsf{CL}_{\mathsf{td}}$ can simulate $\mathsf{CL}$.

## Related Work

Arai, Pitassi, and Urquhart (2001) present a relative efficiency study of variations of analytic tableaux (Smullyan 1968) based on restrictions on the decomposition strategy for formulas (*clausal*, *generalized clausal*, and *binary* tableaux). The effect of adding branching (resulting in the tableau method KE) to analytic tableaux is studied in (D'Agostino and Mondadori 1994). Järvisalo, Junttila, and Niemelä (2005) study the effect of a variety of static (including *input-restricted branching*) and dynamic branching restrictions (including $\mathsf{DPLL}_{\mathsf{td}}$ but excluding $\mathsf{DPLL}_{\mathsf{jf}}$) for $\mathsf{DPLL}$ without clause learning, while Järvisalo and Junttila (2007) study the case of input-restricted branching $\mathsf{CL}$. Finally, Hwang and Mitchell (2005) study typical branching schemes in CSP solving (*2-way* and *d-way branching*).

## References


Arai, N. H.; Pitassi, T.; and Urquhart, A. 2001. The complexity of analytic tableaux. In *STOC*, 356–363. ACM.

Beame, P., and Pitassi, T. 1998. Propositional proof complexity: Past, present, and future. *B. EATCS* 65:66–89.

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res.* 22:319–351.

Ben-Sasson, E.; Impagliazzo, R.; and Wigderson, A. 2004. Near optimal separation of tree-like and general resolution. *Combinatorica* 24(4):585–603.

Cook, S. A. 1976. A short proof of the pigeon hole principle using extended resolution. *SIGACT News* 8(4):28–32.

D'Agostino, M., and Mondadori, M. 1994. The taming of the cut: Classical refutations with analytic cut. *Journal of Logic and Computation* 4(3):285–319.

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. *J. ACM* 7(3):201–215.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem proving. *Communications of the ACM* 5(7):394–397.

Ganai, M. K.; Zhang, L.; Ashar, P.; Gupta, A.; and Malik, S. 2002. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *DAC*, 747–750. ACM.

Haken, A. 1985. The intractability of resolution. *Theoretical Computer Science* 39(2–3):297–308.

Hwang, J., and Mitchell, D. G. 2005. 2-way vs. d-way branching for CSP. In *CP*, volume 3709 of *LNCS*, 343–357. Springer.

Järvisalo, M., and Junttila, T. 2007. Limitations of restricted branching in clause learning. In *CP*, volume 4741 of *LNCS*, 348–363. Springer.

Järvisalo, M.; Junttila, T.; and Niemelä, I. 2005. Unrestricted vs restricted cut in a tableau method for Boolean circuits. *Ann. Math. Artif. Intell.* 44(4):373–399.

Junttila, T. A., and Niemelä, I. 2000. Towards an efficient tableau method for boolean circuit satisfiability checking. In *CL 2000*, volume 1861 of *LNCS*, 553–567. Springer.

Kuehlmann, A.; Paruthi, V.; Krohm, F.; and Ganai, M. K. 2002. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE T-CAD* 21(12):1377–1394.

Kuehlmann, A.; Ganai, M. K.; and Paruthi, V. 2001. Circuit-based Boolean reasoning. In *DAC*, 232–237. ACM.

Lu, F.; Wang, L.-C.; Cheng, K.-T.; and Huang, R. C.-Y. 2003. A circuit SAT solver with signal correlation guided learning. In *DATE*, 892–897. IEEE.

Marques-Silva, J. P., and Sakallah, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48(5):506–521.

Papadimitriou, C. H. 1995. *Computational Complexity*. Addison-Wesley.

Smullyan, R. M. 1968. *First–Order Logic*. Springer.

Thiffault, C.; Bacchus, F.; and Walsh, T. 2004. Solving non-clausal formulas with DPLL search. In *CP*, volume 3258 of *LNCS*, 663–678. Springer.