Using Combinatorial Benchmarks to Probe the Reasoning Power of Pseudo-Boolean Solvers

Jan Elffers[1], Jesús Giráldez-Cru[1], Jakob Nordström[1], and Marc Vinyals[2]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
{elffers,giraldez,jakobn}@kth.se
[2] Tata Institute of Fundamental Research, Mumbai, India
marc.vinyals@tifr.res.in

Abstract. We study cdcl-cuttingplanes, Open-WBO, and Sat4j, three successful solvers from the Pseudo-Boolean Competition 2016, and evaluate them by performing experiments on crafted benchmarks designed to be trivial for the cutting planes (CP) proof system underlying pseudo-Boolean (PB) proof search but yet potentially tricky for PB solvers.
Our experiments demonstrate severe shortcomings in state-of-the-art PB solving techniques. Although our benchmarks have linear-size tree-like CP proofs, and are thus extremely easy in theory, the solvers often perform quite badly even for very small instances. We believe this shows that solvers need to employ stronger rules of cutting planes reasoning. Even some instances that lack not only Boolean but also real-valued solutions are very hard in practice, which indicates that PB solvers need to get better not only at Boolean reasoning but also at linear programming.
Taken together, our results point to several crucial challenges to be overcome in the quest for more efficient pseudo-Boolean solvers, and we expect that a further study of our benchmarks could shed more light on the potential and limitations of current state-of-the-art PB solving.

1 Introduction

In its most general form, a *pseudo-Boolean function* maps sets of Boolean values to a real number. Such functions have been studied since the 1960s in the context of operations research and 0-1 programming, yielding an extensive body of work as surveyed, e.g., in [5]. In this paper we consider the special case of *linear pseudo-Boolean constraints* $\sum_i a_i \ell_i \geq A$ encoded as integer linear combinations of literals ℓ_i (i.e., Boolean variables or negations of such variables). In the decision problem *pseudo-Boolean solving (PBS)* one asks whether a collection of such constraints is feasible or not. In *pseudo-Boolean optimization (PBO)* the task is to compute the best value of an objective function (written as a linear constraint) subject to other linear constraints, a formalism that captures problems in many different fields. Clearly, PBO can be reduced to PBS by iteratively computing solutions and adding constraints forcing the value of the objective function to improve. In the current work we focus on the decision problem PBS.

Pseudo-Boolean constraints are more expressive than conjunctive normal form (CNF) formulas, but are close enough that techniques for CNF SAT solving

can be harnessed to attack pseudo-Boolean problems. The connection to integer linear programming (ILP) and, in particular 0-1 programming, makes it natural to also borrow insights from these areas.

Work on applying SAT-based methods in pseudo-Boolean solving seems to have started in the mid-1990s inspired by Barth [1, 2] and developed in different directions. One line of research has focused on inference methods based on *cutting planes (CP)* [7, 9, 14], including works by Chai and Kuehlman [6], Sheini and Sakallah [33], and Dixon et al. [10]. In this context it was reported that focusing on the restricted form of *cardinality constraints* $\sum_i \ell_i \geq A$ can be more effective than dealing with general linear constraints [6, 33], and according to [11] a very competitive approach can be to simply translate pseudo-Boolean constraints to CNF and use a *conflict-driven clause learning (CDCL)* SAT solver [3, 25, 28]. A different path was pursued by Manquinho and Marques-Silva, who devised ways of learning and backtracking non-chronologically using branch-and-bound search [22, 23]. Needless to say, this brief historical overview is very far from complete—see, e.g., the excellent survey [31] for more details.

Current state-of-the-art pseudo-Boolean solvers building on the first line of work discussed above include *Open-WBO* [26, 29], which reduces the problem to CNF [19] and applies CDCL search, and *Sat4j* [21, 32], which uses cutting planes inference rules. These two solvers performed very well in the decision track in the *Pseudo-Boolean Competition 2016* together with the relatively new solver *cdcl-cuttingplanes* [12],[3] which, as the name suggests, also implements conflict-driven search in cutting planes.

1.1 Our Investigations and Conclusions

The survey [31] mentioned above ends on the optimistic note that *"[t]rade-offs between inference power and inference speed are often made in current algorithms and the right balance is still sought"* but that *"we can expect that, once the right balance is found, pseudo-Boolean solvers will become a major tool in problem solving."* From a theoretical point of view, there are strong reasons to concur with this—pseudo-Boolean (PB) solvers are based on an exponentially stronger method than CDCL solvers and so should have the potential to vastly outperform them. Intriguingly, in practice the opposite more often seems to be the case.

We approach this disconnect between theory and practice by studying the performance of the three PB solvers *cdcl-cuttingplanes*, *Open-WBO*, and *Sat4j* on the kind of PBS decision problems where they came out on top in the Pseudo-Boolean Competition 2016. We consider the benchmarks in [36][4] as well as other

[3] An updated version of this solver with the new name *RoundingSat* is described in [13].

[4] It should be noted that [36] is closely related to the current work in that both papers are motivated by similar concerns, namely understanding the power and limitations of pseudo-Boolean reasoning. A key difference, though, is that the instances studied in [36] are designed to be potentially *hard* for the subsystems of cutting planes implemented by PB solvers, whereas in this paper we choose parameter settings so that almost all instances are theoretically *very easy*.

crafted benchmarks that were specifically designed to be very easy for the cutting planes proof system underlying pseudo-Boolean SAT solving but to be potentially tricky to handle for PB solvers (not in the sense of being "obfuscated" in any way, but in the sense that the instances seem to require inherently pseudo-Boolean reasoning to be efficiently solvable). Since our starting point is proof complexity, our focus is on unsatisfiable benchmarks. We report results from fairly extensive experiments intended to highlight strong and weak points of these solvers when run on our benchmark set, and present some empirical conclusions as well as hypotheses that we hope will stimulate follow-up research.

Before briefly discussing our findings, we want to stress that we do not claim to provide the final word in this matter, but rather our purpose is to initiate a new line of research. By necessity, our set of benchmarks is limited, and the instances are quite particular in that they have been designed to have certain combinatorial properties. Nevertheless, we believe that this work shows that an in-depth study of pseudo-Boolean solver performance on such benchmarks can provide interesting insights. In contrast to industrial benchmarks, here we can have a detailed understanding of the properties of the instances, including, in particular, the fact that they can be solved efficiently in principle. In addition, the possibility to scale their size allows us to draw conclusions about asymptotic behaviour rather than just observing isolated data points.

The need for stronger Boolean reasoning The most obvious conclusion from our work is that the cutting planes-based reasoning in pseudo-Boolean solvers needs to be strengthened significantly. As mentioned above, our benchmarks have been designed to have short CP proofs, and most often these proofs are even tree-like (meaning that they can be found without learning from conflicts). However, in many cases the solvers struggle hopelessly even for very small instances. To explain by an analogy to CDCL solving, this is as if state-of-the-art CDCL solvers would fail completely for small formulas with linear-size DPLL proofs!

We consider the most plausible explanation for the poor performance to be that the PB solvers do not exploit the full power of the *division* rule in cutting planes, using only a limited form of division as in *cdcl-cuttingplanes* or substituting it altogether by the simpler *saturation* rule as in *Sat4j*. This hypothesis is strengthened by the observation that *cdcl-cuttingplanes* is consistently performing better than *Sat4j* in cases when use of the division rule seems to be crucial from a theoretical point of view.

Looking at the results from a different angle, it is well known that PB solvers such as *Sat4j* performs well on *pigeonhole principle (PHP) formulas*, and the results from the Pseudo-Boolean Competition 2016 show that this is also the case for so-called *subset cardinality formulas* [27, 34, 35]. However, the seemingly equally simple *even colouring formulas* [24] appear very hard in practice. On closer inspection, one difference here is that PHP formulas do not have even rational solutions—there is no way to fit $n + 1$ pigeons into n holes even if the pigeons can be sliced—and the same holds for subset cardinality formulas, whereas even colouring formulas are satisfied by assigning every variable value $\frac{1}{2}$.

This raises the question whether hardness correlates with the existence of rational solutions, which we will refer to in what follows as the *rational hypothesis*. Clearly, rational solutions alone do not imply hardness—any 2-CNF formula without unit clauses is satisfied by assigning all variables value $\frac{1}{2}$, yet this does not make such a formula hard—but any formula *without* rational solutions has short proofs that PB solvers can find in theory [36], so we can ask if they can also find such proofs in practice.

Although there are families of formulas where the lack of rational solutions seems to help, the answer from our experiments to whether solvers can *always* efficiently decide rationally infeasible 0-1 integer linear programs is negative—we find examples of instances without rational solutions that are very hard in practice. But when we then go further and study for which instances we can help the solvers to run fast by, e.g., dropping a heavy hint in the form of a good fixed variable order (while keeping other settings at default values), an intriguing pattern emerges—for most of our benchmarks it holds that the solvers can be made efficient *if the instances have small strong backdoors*[5] *to pseudo-Boolean formulas without rational solutions.* There is of course a selection bias in the benchmarks we study, but we nevertheless find this refined version of the rational hypothesis quite intriguing and hope it can stimulate further study.

The need for stronger LP reasoning The refined rational hypothesis just discussed cannot explain all our findings, however, especially since solvers cannot always count on getting helpful hints. For some of our benchmark families—in particular, encodings of the *dominating set* problem on hexagonal grids—the formulas are extremely challenging even when the corresponding linear program has no rational solutions.[6] For these instances it can be shown that the method of reasoning used in *Sat4j* and *cdcl-cuttingplanes* can in principle derive extra constraints by simple addition [36], i.e., without any Boolean reasoning, and with these constraints added the formulas become trivial also in practice. The solvers do not find these constraints on their own, though, and we have not been able to coax them into doing so even by trying out different helpful variable orderings. Instead, the solvers get stuck exploring parts of the search space where even the LP is infeasible. This shows that PB solvers need to strengthen not only their Boolean reasoning but also their linear programming capabilities.

The need to become more competitive with CDCL and MIP solvers For formulas that are provably exponentially hard for resolution but easy for cutting planes we see, not surprisingly, that *cdcl-cuttingplanes* and *Sat4j* outperform *Open-WBO*. However, for instances that are inherently pseudo-Boolean in nature, but where resolution can nevertheless efficiently simulate PB reasoning if given a natural

[5] A *strong backdoor* for an instance F to a family $\mathcal{F}$ of (easy) instances—in this case, instances without rational solutions—is a set of variables in F such that any assignment ρ to these variables yield a restricted instance $F\restriction_\rho$ that is in $\mathcal{F}$.

[6] It might be worth pointing out that for an instance to lack rational solutions is the same as saying that the linear programming relaxation is infeasible, and so such instances can be shown unsatisfiable in polynomial time simply by solving the LP.

CNF encoding, we see that most often *cdcl-cuttingplanes* and *Sat4j* are orders of magnitude slower than *Open-WBO*. It is also often the case, though, that the roles become reversed if the formula is randomly shuffled before being fed to the solvers. And it is also often true that if we force *cdcl-cuttingplanes* to use a good fixed decision order, then its performance matches that of *Open-WBO*, but if left to its own devices *cdcl-cuttingplanes* will deviate from this decision order. This raises the question of whether the way *Open-WBO* encodes pseudo-Boolean constraints into CNF helps it to find and stick to a good variable order when the formula is presented in such a way as to suggest such a good order.

Since pseudo-Boolean solving is closely related to integer linear programming, it is also natural to compare PB solvers to mixed integer linear (MIP) solvers. We have run experiments with the MIP solver *Gurobi* [15] on our combinatorial benchmarks and can observe that it is consistently better than all the PB solvers studied. It should be emphasized that this is perhaps not so suprising—many of our formulas have been constructed to be hard for CDCL but trivial for tree-like cutting planes, and this means that they are by definition amenable to branch-and-bound techniques. Furthermore, *Gurobi* solves an LP relaxation at every node in its search tree, and so will immediately detect the rationally infeasible instances that turn out to be hard for PB solvers. Thus, for the benchmarks considered in this paper *Gurobi* is playing on home turf. Still, we can see no good reason why PB solvers should be so bad for formulas that are dead-easy for tree-like CP. And it certainly would seem well worth it to take a long, hard look at MIP solving techniques and see what can be ported to PB solvers.

Our findings might seem depressing in that they are mostly bad news for state-of-the-art pseudo-Boolean solving. However, we would rather view our work as pointing forward to some crucial challenges that need to be overcome. We hope that our combinatorial formulas can be valuable as challenge benchmarks in the quest to develop more efficient PB solvers, which could then fulfil the vision of [31] and assume their rightful place as *"major tools in problem solving."*

1.2 Organization of This Paper

We describe our experimental set-up in Section 2 and discuss our benchmarks in Section 3. Section 4 contains an analysis of our results, and some concluding remarks are presented in Section 5.

2 Experimental Set-up

We have conducted an experimental evaluation using the pseudo-Boolean solvers *cdcl-cuttingplanes* [12] *Open-WBO* [26, 29], and *Sat4j* [21, 32]. These were the top three solvers in the Pseudo-Boolean Competition 2016 [30] in the category DEC-SMALLINT-LIN ("no optimization, small integers, linear constraints, SAT+UNSAT answers"), and we ran the solvers on such benchmarks as described in more detail in Section 3. Since our benchmarks are inspired by proof complexity, where one studies the complexity of certifying unsatisfiability, we focused almost

exclusively on UNSAT instances. For our experiments on shuffled instances we randomly shuffled the variable indices, literal polarities, and the order of the constraints as well as variables within the constraints.

We used the versions of *cdcl-cuttingplanes* and *Open-WBO* submitted to the PB Competition 2016 and a slightly later version of *Sat4j* from November 4, 2016. By default *Sat4j* runs two subsolvers in parallel (Resolution and CuttingPlanes) and returns the answer of the first of them solving the problem. This gives *Sat4j* an advantage, since it gets double the amount of CPU time compared to the other solvers, but it only makes our point stronger when it performs poorly. Since we are particularly interested in analysing cutting planes-based solvers we included the standalone solver *Sat4jCP* in our experiments, but we only show results when they differ from *Sat4j* (i.e., when the formula was decided by the Resolution subsolver). For the *cdcl-cuttingplanes* experiments with fixed decision orders we used a version from April 19, 2017, since the competition version had no support for fixed orders. For our mixed integer programming experiments we used the solver *Gurobi* [15] version 7.5.2 restricted to a single thread.

We ran our experiments on a cluster with a set-up of 6 AMD Opteron 6238 (2.6 GHz) cores and 16 GB of memory. The timeout for the experiments was 3000 seconds unless otherwise stated.

3 Description of Benchmarks

All our benchmarks were designed to be very easy for the cutting planes (CP) proof system, so that the experiments would measure proof search quality for instances where CP-based solvers should in principle be able to perform well.

The well-known *pigeonhole principle (PHP) formula* claims that $n+1$ pigeons can be placed into n holes with only one pigeon per hole, encoded as *pigeon axioms* $\sum_{j\in[n]} x_{i,j} \geq 1$ and *hole axioms* $\sum_{i\in[n+1]} x_{i,j} \leq 1$ for $i \in [n+1]$, $j \in [n]$. We also consider more complicated versions by introducing *emergency exits* as follows. We generate k disjoint PHP instances over variables $x^{\ell}_{i,j}$, where for each $\ell \in [k]$ we allow some pigeon(s) i^* to "take the emergency exit" by changing the pigeon axiom to $y^{\ell} + \sum_{j\in[n]} x^{\ell}_{i^*,j} \geq 1$, where y^{ℓ}, $\ell \in [k]$, are new variables. However, a special constraint $\sum_{\ell=1}^{k} y^{\ell} \leq k-1$ enforces that at most $k-1$ emergency exits are taken in total. We study two variants where either (a) one particular pigeon per PHP instance can take the emergency exit or (b) all pigeons in an instance can do so. All these versions of PHP are rationally unsatisfiable.

A *subset cardinality (SC) formula* [27, 34, 35] is generated from a 0/1 $n \times n$ matrix $A = (a_{i,j})$ with 4 ones in every row and column, except that one row and column contains 5 ones. Writing $R_i = \{j \mid a_{i,j} = 1\}$ and $C_j = \{i \mid a_{i,j} = 1\}$ to denote the positions of 1s in row i and column j, respectively, the formula obtained from A consists of the constraints $\sum_{j\in R_i} x_{i,j} \geq |R_i|/2$ and $\sum_{i\in C_j} x_{i,j} \leq |C_i|/2$ for $i, j \in [n]$. We use randomly generated matrices and "fixed bandwidth" matrices with a fixed pattern of ones shifted down the diagonal. We also consider a more restrictive version with equality constraints $\sum_{j\in R_i} x_{i,j} = \lceil |R_i|/2 \rceil$ and $\sum_{i\in C_j} x_{i,j} = \lfloor |C_i|/2 \rfloor$. Again, all of these instances are rationally unsatisfiable.

The *even colouring (EC) formula* [24] over a connected graph $G = (V, E)$ with all $v \in V$ of even degree $\deg(v)$ consists of constraints $\sum_{e \in E(v)} x_e = \deg(v)/2$, where $E(v)$ denotes the set of edges incident to v. This formula claims the existence of a black-white edge colouring such that every vertex has the same number of black and white edges, and is unsatisfiable if and only if $|E|$ is odd. We study these formulas for two families of graphs: (a) long, narrow toroidal grids, where every vertex has edges horizontally and vertically to its 4 neighbours, and with one edge split into a degree-2 vertex to get an odd number of edges; (b) random regular graphs of even degree $2d$, splitting an edge if d is even.

The *vertex cover (VC) formula* with constraints $x_u + x_v \geq 1$, $(u, v) \in E$, and $\sum_{v \in V} x_v \leq S$ encodes that a graph $G = (V, E)$ has a size-S vertex cover (i.e., a set such that every edge is incident to some vertex in it). As in [36], we examine long, narrow rectangular toroidal grids with m rows and n columns for $m = \mathrm{O}(1)$ even. The minimal vertex cover for such a graph has size $m\lceil n/2 \rceil$. We generate four versions by varying the value of S, where for the first three n is odd: (a) $S = m\lceil n/2 \rceil - 1$, the largest value such that the formula is still unsatisfiable (version *hard*); (b) $S = mn/2$ (version *easy*), which is more obviously unsatisfiable but still has a rational solution with value $\frac{1}{2}$ for all variables; (c) $S = m\lfloor n/2 \rfloor - 1$ (version *norat*), without rational solutions; (d) $S = m\lfloor n/2 \rfloor - 1$ for n even (version *norat-even*), where S is the largest value that makes the formula unsatisfiable both for Boolean and rational values. To obtain slightly harder instances without superfluous edges we also consider such grids with all vertical edges removed, yielding a collection of disjoint cycles, and use the same values of S as above.

The *dominating set (DS) formula* for a graph G consists of constraints $\sum_{u \in \{v\} \cup N(v)} x_u \geq 1$ for all $v \in V$ and $\sum_{v \in V} x_v \leq S$, saying that G has a size-S dominating set (i.e., such that every vertex in G either belongs to or is a neighbour of a vertex in the set). We study these formulas for hexagonal grids as represented in [36] with m rows and n columns, with one dimension fixed while the other scales. Since hexagonal grids are 3-regular any dominating set must have size at least $\lceil |V|/4 \rceil$, and so we choose $S = \lfloor |V|/4 \rfloor$. When $4 \nmid |V|$ the resulting instance is rationally unsatisfiable, but otherwise there is always a rational solution setting all variables to $\frac{1}{4}$, whereas the Boolean satisfiability depends in nontrivial ways on the exact geometry of the grid [36] (in particular, in contrast to the other families some of our dominating set instances are satisfiable).

The *linearized pebbling (LinPeb) formula* of arity d over a directed acyclic graph with a unique sink has variables $v_1, \ldots, v_d$ for each vertex v and consists of the following contradictory constraints (where we let $d' = 2\lfloor (d-1)/2 \rfloor + 1$): (a) for every source vertex v the constraint $2\sum_{i=1}^{d} v_i \geq d'$; (b) for every non-source vertex w with predecessors u and v the constraint $2\sum_{i=1}^{d} w_i \geq \sum_{i=1}^{d} (u_i + v_i)$; (c) for the unique sink vertex z the constraint $2\sum_{i=1}^{d} z_i \leq d'$. We study instances generated from so-called pyramid graphs.[7]

[7] We remark that some linearized pebbling formulas were submitted to the Pseudo-Boolean Competition 2016 under the name `sumineq` (sum inequalities).

Table 1: Overview of benchmarks and results

Formula family	Rational solutions	Small backdoor	Division needed	Performance cdcl-CP	Sat4j	O-WBO
PHP	No	–	–	Easy	Easy	Hard
SC	No	–	–	Easy	Easy	Hard
EC even grid	Yes	Yes	No	Easy	Easy	Hard
EC odd grid	Yes	No	Helpful	Hard[a]	Hard	Hard
EC random	Yes	No(?)	Crucial(?)	Fairly hard	Hard	Hard
VC hard	Yes	No	Helpful	Hard[b]	Hard	Easy[c]
VC easy	Yes	Many	No	Hard[b]	Hard	Easy[c]
VC norat(-even)	No	–	–	Hard[b]	Hard	Easy[c]
DS	Yes	Many	No	Hard[d]	Hard[d]	Easy[c]
LinPeb	Yes	Yes	No	Hard[b]	Hard[e]	Easy

[a] Easy if formula appropriately reordered. [b] Fairly easy for forced order.
[c] Hard if shuffled. [d] Easy if LP-derivable constraints added. [e] Easy for *Sat4jRes*.

4 Experimental Evaluation

In this section we describe and analyse the results of our experiments.[8] As already mentioned, our benchmarks can be scaled in size by varying a parameter, allowing to study the asymptotic behaviour of the solvers as the instance size increases. Our figures illustrate this by plotting performance on the y-axis against the value of the scaling parameter on the x-axis (which, in particular, seems to be a better way of visualizing our data than using so-called cactus plots).

4.1 Pseudo-Boolean Solvers and Boolean Reasoning

Our first conclusion is that PB solvers need to strengthen their reasoning by using stronger rules than saturation and implementing better proof search.

As an example, consider even colouring (EC) formulas, which can be refuted (i.e., proven unsatisfiable) with tree-like cutting planes proofs in linear size using just two applications of division. We could thus expect solvers based on cutting planes (CP) to run blisteringly fast for EC formulas over any graph, but this is not the case. We have generated formulas from $m \times n$ grids with $m = \mathrm{O}(1)$ a small constant, which have short proofs in resolution when encoded in CNF, and random regular graphs, which are exponentially hard for resolution.[9]

For grids with m even the formulas are trivial for both *Sat4j* and *cdcl-cuttingplanes*, but for m odd they are hard. Interestingly, in the latter case *Sat4jRes* performs better than *Sat4jCP*, suggesting that CP-based solvers do not

[8] By necessity, our discussion is far from exhaustive, but readers can find all our benchmarks and the data from our experiments at www.csc.kth.se/~jakobn/publications/CombinatorialBenchmarksPBsolvers.

[9] Such a lower bound cannot be found in the literature, but is possible to obtain for graphs with good enough expansion using a variation of the techniques in [4].

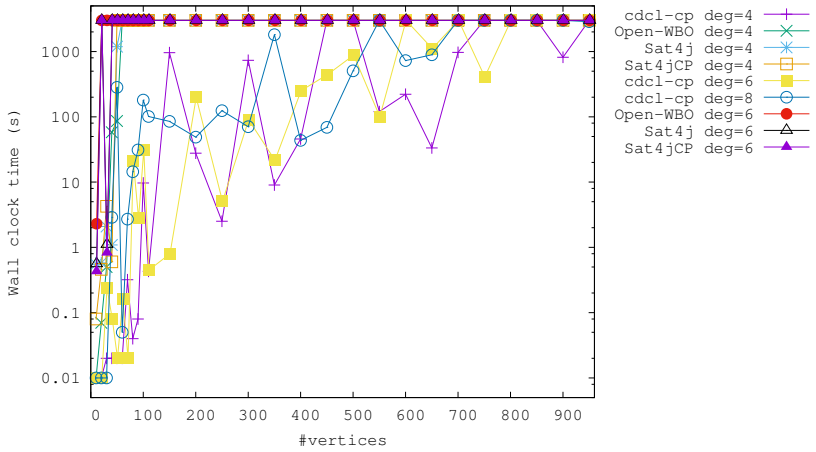

Fig. 1: Solver performance for even colouring formulas on random graphs (#constraints =#vertices $|V|$; #variables = $\mathtt{deg} \cdot |V|/2 + \mathrm{O}(1)$).

find better proofs than CDCL-based solvers. It is also notable that flipping the vertex (and hence variable) order from column-major to row-major, even though it does not change the formula, helps *cdcl-cuttingplanes* find an efficient proof. This indicates that there is ample room to improve on search heuristics.

To explain the difference between even-row and odd-row grids, we observe that EC formulas always have a rational solution with all variables assigned value $\frac{1}{2}$, but grids with an even number of rows and columns have backdoors of size 1 (namely, either of the edges incident to the degree-2 vertex on the split edge), and as long as the number of rows m is even there are backdoors of size at most $m = \mathrm{O}(1)$. If m is odd, however, the backdoor size jumps to $n - \mathrm{O}(1)$.

EC formulas on random graphs are very hard for *Open-WBO* and *Sat4j*. They are not easy for *cdcl-cuttingplanes* either, but this solver performs markedly better, and does not seem to be sensitive to the degree of the graph (see Figure 1). The only short proofs known for these formulas crucially use division [36], and we believe that the superior performance of *cdcl-cuttingplanes* is explained by the frequent (though still limited) use of division in this solver. It is worth noting, though, that for the few instances solved by *Sat4j* the number of conflicts is not too far from *cdcl-cuttingplanes*. The most likely explanation is that *Sat4j* does divide the constraint in the rare case when all coefficients are equal, which is all that is needed in [36]. To be sure whether the above explanations are correct we would need to do proof logging, but for PB solvers there is unfortunately nothing like the *DRAT* format [17, 18, 37] used for CDCL solvers.

Another formula family where *cdcl-cuttingplanes* performs better than *Sat4j* are linearized pebbling formulas, which are easy for *Sat4jRes* but extremely hard for *Sat4jCP* (see Figure 2). Interestingly, for these instances *Sat4j* generates constraints with coefficients larger than 10^9 in a matter of seconds, whereas

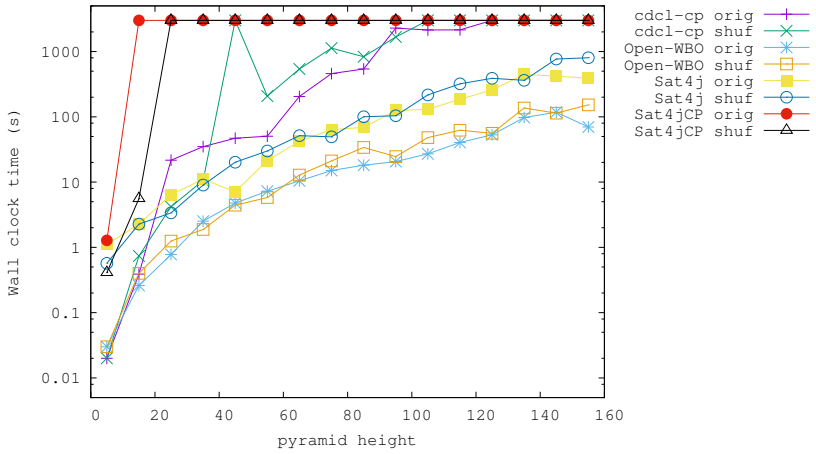

Fig. 2: Performance for arity-5 linearized pebbling formulas on pyramids, with (`shuf`) and without (`orig`) shuffling (#constraints $\approx$ #variables $\approx$ `height`$^2/2$).

cdcl-cuttingplanes keeps all coefficients small. Division is not critically needed for efficient refutations here, but it might be that it is what helps *cdcl-cuttingplanes* keep coefficient sizes down and achieve better performance. *Sat4j* has problems with large coefficients also for vertex cover (VC) and dominating set (DS) instances, where *cdcl-cuttingplanes* performs better, but not for PHP and subset cardinality (SC) formulas, where both solvers are fast.

Let us next review what our data say about the *extended rational hypothesis*, i.e., that instances with small backdoors to rational unsatisfiability should be easy. PHP and SC formulas do not have rational solutions, and the fact that instances are trivial for both *cdcl-cuttingplanes* and *Sat4j* supports the rational hypothesis. PHP formulas with emergency exits were designed to be potentially harder instances that still do not have rational solutions, but they fail to fool *cdcl-cuttingplanes* and hence can be interpreted as circumstantial evidence in favour of the rational hypothesis for this solver (but less so for *Sat4j*).

As mentioned above, *cdcl-cuttingplanes* and *Sat4j* run fast on EC formulas for backdoor size 1 but not larger (random graphs very likely yield instances without small backdoors, though we did not attempt a rigorous proof). This supports the hypothesis, but the fact that *cdcl-cuttingplanes* also runs fast when slightly modifying instances for odd-row grids makes the connection less clear.

The VC instances `norat` and `norat-even` without rational solutions are easier than the `easy` version, which is in turn easier than the `hard` version. This is consistent with the hypothesis since the backdoor size is 0 for `norat(-even)` and 1 for `easy` (rational solutions disappear after branching on any vertex), whereas the smallest backdoor for version `hard` has size $m-1$ ($m-1$ vertices in the same column form a backdoor, but any $m-2$ vertices can be assigned to leave a rational solution). This holds for both grids and collections of cycles.

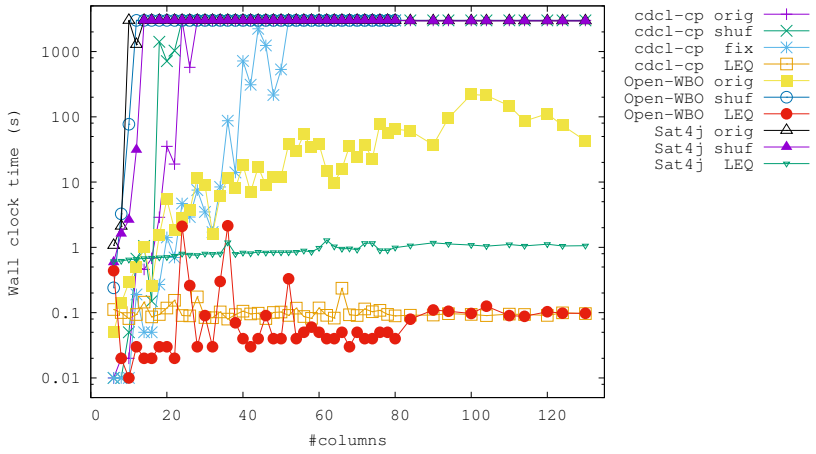

Fig. 3: Performance for dominating set on hex grids with 7 rows with shuffling (`shuf`) and without (`orig`) and also with added `LEQ` constraints as well as for fixed-order (`fix`) *cdcl-cuttingplanes* (#constraints = #variables = $7 \cdot \texttt{columns} + \mathrm{O}(1)$).

Linearized pebbling formulas have a size-d backdoor (the d variables associated with the sink). *Sat4jCP* does not run fast on these formulas, but with some tweaking *cdcl-cuttingplanes* can be convinced to perform well. It seems like a stimulating challenge to develop natural heuristics for the CP-based solvers that would make them competitive with *Open-WBO* and *Sat4jRes* for these instances.

Dominating set formulas on hexagonal grids have rational solutions when the total number of vertices is divisible by 4, in which case there is strong empirical evidence for backdoors of size 3 (obtained by considering any vertex and two of its neighbours). Somewhat annoyingly, we have not been able to always make *cdcl-cuttingplanes* run fast for such instances, however, so as of now our experimental results for these formulas do *not* support the rational hypothesis.

4.2 Pseudo-Boolean Solvers and Linear Programming

To support the claim that PB solvers also need better linear programming capabilities, we again consider dominating set instances on hexagonal grids. These are very challenging for both *Sat4j* and *cdcl-cuttingplanes*. They are manageable for *Open-WBO* when the fixed dimension is small but quickly become very hard as this dimension grows. Also, *Open-WBO* is extremely sensitive to random shuffling, a phenomenon that we discuss further in Section 4.3.

Quite intriguingly, all instances become trivial if modified as follows. Recall that we have a greater-or-equal (`GEQ`) constraint $\sum_{u \in \{v\} \cup N(v)} x_u \geq 1$ for each vertex v encoding that it is dominated. Since hexagonal grids are 3-regular, and since the required dominating set size is at most $|V|/4$, it follows that at most one of the vertices in $\{v\} \cup N(v)$ is in the dominating set. These less-or-equal (`LEQ`)

constraints can easily be derived using only the addition rule in *cdcl-cuttingplanes* and *Sat4j* [36], and with such constraints added the instances become trivial as shown in Figure 3. So far we have not been able to get the solvers to realize that these `LEQ` constraints should be derived, though, although elementary linear programming would be sufficient to achieve this.

4.3 Pseudo-Boolean Solvers Versus CDCL and MIP

When comparing cutting planes-based solvers to CDCL solvers we get mixed results. On PHP and subset cardinality formulas both CP-based solvers *Sat4j* and *cdcl-cuttingplanes* perform very well, while *Open-WBO* does very poorly, which nicely matches that these formulas are easy for CP but exponentially hard for resolution. For our other benchmarks we find *Open-WBO* to be surprisingly competitive, but performance is often brittle. In contrast, although all our benchmarks are very easy for CP, on many instances *cdcl-cuttingplanes* and *Sat4j* are quite far from performing well (though *cdcl-cuttingplanes* can often be made to match *Open-WBO* performance by manual intervention such as fixing good variable decision orders).

An example family of benchmarks for which *Open-WBO* shines are vertex cover (VC) formulas. Here the performance of *cdcl-cuttingplanes* and *Sat4j* is quite poor in general, though the former solver is clearly better than the latter. A closer look at the results reveals that the number of conflicts seems to be similar, but since *Sat4j* solves so few instances within the timeout limit it is hard to know for sure whether the differences in running time are due to proof search quality or lower-level implementation details.

Open-WBO performs quite well for almost all our VC instances (except for the `hard` version on collections of cycles), which indicates that the encoding to CNF admits an efficient resolution proof. Since the covering constraints $x_u + x_v \geq 1$ for the edges $(u, v) \in E$ are already disjunctive clauses, the performance is likely to depend on how the vertex cover size constraint $\sum_{v \in V} x_v \leq S$ is encoded into CNF. A key aspect here is that the vertices are listed consecutively when we generate the grid graphs (more precisely, in column-major order). This means that as *Open-WBO* decides on consecutive variable indices, it will explore neighbouring vertices constrained by common covering constraints, and it seems plausible that propagation on the auxiliary variables in the encoding of the size constraint helps the solver count efficiently. This hypothesis[10] is supported by the fact that when instances are shuffled the performance degenerates dramatically for *Open-WBO*, but much less so for *cdcl-cuttingplanes*. A further observation is that though *cdcl-cuttingplanes* is rather bad for these instances, it can be made much more efficient by fixing the decison order (namely, branching on vertices in column-major order). This indicates that with a better decision heuristic *cdcl-cuttingplanes* could potentially be competitive with *Open-WBO* here. See Figure 4 for plots of all the findings above.

[10] It would be interesting to verify this by a more in-depth study of *Open-WBO*. However, the PB version of this solver was not open-source at the time of our experiments, and also our main focus in this work is on solvers implementing CP-based reasoning.

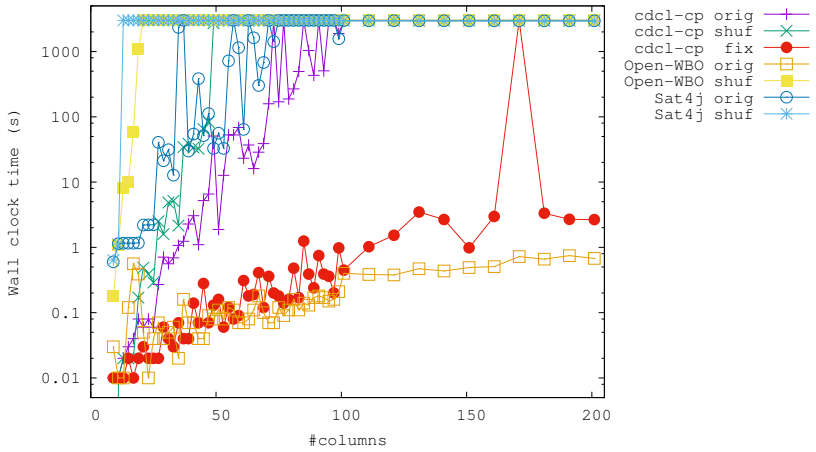

Fig. 4: Performance for vertex cover formulas on grids with 8 rows, version `norat`, with shuffling (`shuf`) and without (`orig`), and also for fixed-order (`fix`) *cdcl-cuttingplanes* (#variables $= |V| = 8 \cdot$ `columns`, #constraints $= 2 \cdot |V|$).

As an example of benchmarks that *Sat4j* and *cdcl-cuttingplanes* can solve easily we have pigeonhole principle (PHP) formulas with $n+1$ pigeons and n pigeonholes. These have CP proofs in size $\mathrm{O}(n^2)$ that can be found with $\mathrm{O}(n)$ conflicts. While both solvers only need $\mathrm{O}(n)$ conflicts, and seem to find the same (essentially optimal) proof, we found that running times scale very differently. A linear regression analysis using the logarithm of the number of constraints indicates running time $\mathrm{O}(n^{3.2})$ for *cdcl-cuttingplanes* but $\mathrm{O}(n^{5.0})$ for *Sat4j* (see Figure 5). Interestingly, this turned out to be due to an implementation inefficiency in *Sat4j*, which could be identified and fixed thanks to our experiments, after which running times became more similar. It is not surprising that PHP formulas are very hard for *Open-WBO*, since there is an exponential lower bound for resolution [16] which can also be adapted (using techniques in [4]) to work for other common ways of encoding the at-most-1 pigeonhole constraints into CNF.

PHP formulas with emergency exits are always easy for *cdcl-cuttingplanes* but remain hard for *Open-WBO* independently of the number of emergency exits k. This latter finding is also as expected, since even if the solver chooses $k-1$ emergency exits in the right way to satisfy $k-1$ subinstances of PHP, the residual formula is a standard PHP instance which is exponentially hard. Interestingly, *Sat4j* performs well on the version where all pigeons can take the emergency exit, but much worse on the version with only one pigeon per exit (which is more constrained, and could thus have been expected to be easier). We remark that when both *Sat4j* and *cdcl-cuttingplanes* solve these formulas efficiently the number of conflicts seem to grow like $\mathrm{O}(kn)$, but when *Sat4j* does not perform well the number of conflicts grows faster. Thus, in contrast to the results for standard PHP, here the proof search quality seems worse in *Sat4j*.

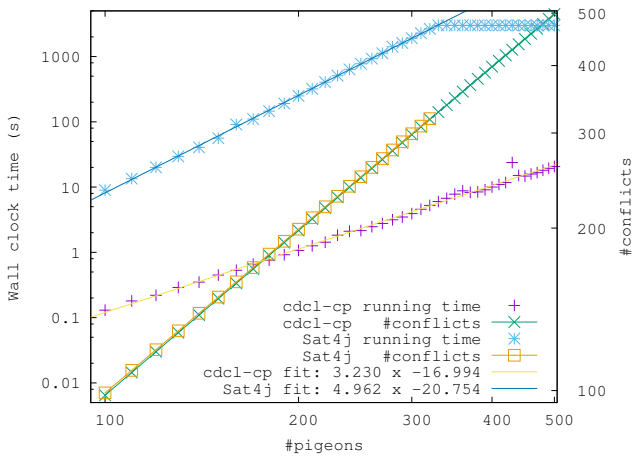

Fig. 5: Running time and #conflicts for *cdcl-cuttingplanes* and *Sat4j* on PHP (#variables = $\texttt{pigeons}(\texttt{pigeons}-1)$, #constraints = $2\cdot\texttt{pigeons}-1$).

Looking at subset cardinality formulas, they seem to be solved much faster than PHP when plotting running time against the scaling parameter, but this is since the instances are much smaller. Again, we observe that *Sat4j* takes significantly longer than *cdcl-cuttingplanes* as the instance size increases. *Open-WBO* is completely lost, as expected in view of the exponential lower bound for resolution in [27] (see Figure 6).

Let us finally make a brief comparison to MIP solving. *Gurobi* does remarkably well on all our benchmarks, solving all but the three largest EC instances in under 10 seconds. On the one hand, this is not too surprising, since all of our instances have tree-like proofs, and hence just branching and backtracking without learning is enough to solve them. Furthermore, the challenging instances that are rationally unsatisfiable will be solved *Gurobi* very quickly, since it also considers linear relaxations of the problem and this is enough to decide unsatisfiability. However, it is hard to avoid the conclusion that one promising approach for strengthening PB solvers would be to incorporate techniques from MIP solving.

5 Concluding Remarks

In this paper we evaluate the three pseudo-Boolean solvers *cdcl-cuttingplanes*, *Open-WBO*, and *Sat4j* on decision problems encoded as linear constraints with small integer coefficients, a kind of problems where these solvers were among the best in the Pseudo-Boolean Competition 2016. The solvers differ in that *Open-WBO* re-encodes the problem into CNF and runs a CDCL solver, thus performing proof search in resolution for the re-encoded instance, whereas *cdcl-cuttingplanes* and *Sat4j* implement conflict-driven search natively with pseudo-Boolean constraints, corresponding to cutting planes (CP) proof search.

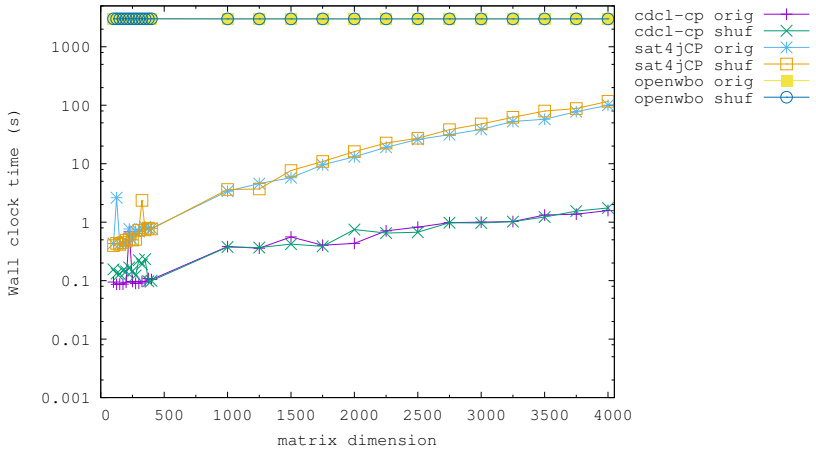

Fig. 6: Performance for subset cardinality formulas on random graphs (#variables $= 4 \cdot \texttt{dimension} + 1$, #constraints $= 2 \cdot \texttt{dimension}$).

We have performed extensive experiments on carefully constructed combinatorial benchmarks to investigate how efficiently these solvers implement their chosen methods of reasoning. Although all of our instances have been specifically designed to be very easy for the cutting planes proof system, the performance of *cdcl-cuttingplanes* and *Sat4j* varies greatly, and is often quite poor. Theoretical as well as empirical evidence points to the conclusion that the reasoning in these solvers needs to be strengthened, in particular, by exploiting the division rule.

For many of the benchmarks studied we can help *Sat4j* and *cdcl-cuttingplanes* run fast by giving advice in the form of a good, fixed variable decision order, or sometimes by reordering variables and constraints. An immediate question is whether the solvers could achieve such good performance on their own by some enhanced heuristic. It can be observed that this phenomenon occurs most often for instances which either do not even have rational solutions—i.e., when the real polytope defined by the linear constraints is in fact empty—or have small backdoor sets such that any assignment to these backdoor variables eliminates all rational solutions. We find this to be a very intriguing connection, and believe it would be interesting to investigate further whether it can be the case more generally that strong solver performance correlates with the existence of small backdoors to rationally unsatisfiable instances.

As expected, *Open-WBO* stands no chance against *cdcl-cuttingplanes* and *Sat4j* when run on instances that are hard for resolution when encoded into CNF. However, when there are efficient proofs in both resolution and cutting planes we see that the CP-based solvers can be orders of magnitude slower. Curiously, if *cdcl-cuttingplanes* is helped by being given a good variable order on such instances, then the performance is competitive with *Open-WBO*, but when left to its own devices *cdcl-cuttingplanes* does not choose this order. This raises the

question whether the encoding to CNF that is used helps *Open-WBO* find a good variable order and stick with it. It should be noted, though, that *Open-WBO* is very sensitive to permutations of the input, so the encoding to CNF is only good when the constraints in the initial pseudo-Boolean instance are presented in a helpful order. The CP-based solvers appear much more robust in this regard.

Finally, we observe that for the instances considered in this paper all three PB solvers that we study are clearly outperformed by the general-purpose mixed integer programming solver *Gurobi*. At first sight this is slightly disappointing, since PB solvers working on 0/1-valued problems should be able to exploit techniques not available to MIP solvers, but a big part of the explanation is probably that our benchmarks have been constructed to be easy for tree-like CP, and so they are by design amenable to branch-and-bound techniques. But another reason is likely to be that *Gurobi* solves linear programming relaxations of the problem during the search, which makes it run fast on instances that lack rational solutions but are apparently very challenging for PB solvers. We believe that there would be great potential for improvement by incorporating such linear programming reasoning in PB solvers. It would also be interesting to find benchmarks that are easy for conflict-driven pseudo-Boolean search, at least in theory, but not for MIP or CDCL, i.e., instances that are hard for resolution and tree-like cutting planes but easy for general, DAG-like cutting planes.

Taken together, our results can be viewed as a concrete set of challenges to be overcome in order to construct more efficient pseudo-Boolean solvers. It is also our belief that a further study of crafted benchmarks like the ones in this paper has the potential to shed valuable light on the inner workings of PB solvers.

Acknowledgements

We are most grateful to Daniel Le Berre for long and patient explanations of the inner workings of pseudo-Boolean solvers, and to João Marques-Silva for helping us get an overview of relevant references for pseudo-Boolean solving. We also want to thank Ruben Martins for sharing an executable of *Open-WBO* with us and answering questions about the solver. Finally, we are thankful for the many detailed comments from the *SAT 2018* anonymous reviewers, which helped to improve this paper considerably.

Our computational experiments were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC). Many of our benchmarks were generated using the tool `CNFgen` [8, 20], for which we gratefully acknowledge Massimo Lauria.

The fourth author performed part of this work while at KTH Royal Institute of Technology. All authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 279611. The third author was also supported by Swedish Research Council grants 621-2012-5645 and 2016-00782, and the fourth author by the Prof. R Narasimhan Foundation.

References

[1] Barth, P.: Linear 0-1 inequalities and extended clauses. Technical Report MPI-I-94-216, Max-Planck-Institut für Informatik (Apr 1994), preliminary version in *LPAR '93*

[2] Barth, P.: A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik (Jan 1995)

[3] Bayardo Jr., R.J., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97). pp. 203–208 (Jul 1997)

[4] Ben-Sasson, E., Wigderson, A.: Short proofs are narrow—resolution made simple. Journal of the ACM **48**(2), 149–169 (Mar 2001), preliminary version in *STOC '99*

[5] Boros, E., Hammer, P.L.: Pseudo-Boolean optimization. Discrete Applied Mathematics **123**(1–3), 155–225 (Nov 2002)

[6] Chai, D., Kuehlmann, A.: A fast pseudo-Boolean constraint solver. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **24**(3), 305–317 (Mar 2005), preliminary version in *DAC '03*

[7] Chvátal, V.: Edmonds polytopes and a hierarchy of combinatorial problems. Discrete Mathematics **4**(1), 305–337 (1973)

[8] CNFgen: Combinatorial benchmarks for SAT solvers. https://github.com/MassimoLauria/cnfgen

[9] Cook, W., Coullard, C.R., Turán, G.: On the complexity of cutting-plane proofs. Discrete Applied Mathematics **18**(1), 25–38 (Nov 1987)

[10] Dixon, H.E., Ginsberg, M.L., Hofer, D.K., Luks, E.M., Parkes, A.J.: Generalizing Boolean satisfiability III: Implementation. Journal of Artificial Intelligence Research **23**, 441–531 (2005)

[11] Eén, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation **2**(1-4), 1–26 (2006)

[12] Elffers, J.: cdcl-cuttingplanes: A conflict-driven pseudo-Boolean solver (2016), submitted to the *Pseudo-Boolean Competition 2016*

[13] Elffers, J., Nordström, J.: Divide and conquer: Towards faster pseudo-Boolean solving. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-ECAI '18) (Jul 2018), to appear

[14] Gomory, R.E.: An algorithm for integer solutions of linear programs. In: Graves, R., Wolfe, P. (eds.) Recent Advances in Mathematical Programming, pp. 269–302. McGraw-Hill, New York (1963)

[15] Gurobi optimizer. http://www.gurobi.com/

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

[17] Heule, M., Hunt Jr., W.A., Wetzler, N.: Trimming while checking clausal proofs. In: Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13). pp. 181–188 (Oct 2013)

[18] Heule, M., Hunt Jr., W.A., Wetzler, N.: Verifying refutations with extended resolution. In: Proceedings of the 24th International Conference on Automated Deduction (CADE-24). Lecture Notes in Computer Science, vol. 7898, pp. 345–359. Springer (Jun 2013)

[19] Joshi, S., Martins, R., Manquinho, V.M.: Generalized totalizer encoding for pseudo-Boolean constraints. In: Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP '15). Lecture Notes in Computer Science, vol. 9255, pp. 200–209. Springer (August-September 2015)

[20] Lauria, M., Elffers, J., Nordström, J., Vinyals, M.: CNFgen: A generator of crafted benchmarks. In: Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17). Lecture Notes in Computer Science, vol. 10491, pp. 464–473. Springer (Aug 2017)
[21] Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation **7**, 59–64 (2010)
[22] Manquinho, V.M., Marques-Silva, J.: On using cutting planes in pseudo-Boolean optimization. Journal on Satisfiability, Boolean Modeling and Computation **2**, 209–219 (2006), preliminary version in *SAT '05*
[23] Manquinho, V.M., Marques-Silva, J.P.: Integration of lower bound estimates in pseudo-Boolean optimization. In: 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '04). pp. 742–748 (Nov 2004)
[24] Markström, K.: Locality and hard SAT-instances. Journal on Satisfiability, Boolean Modeling and Computation **2**(1-4), 221–227 (2006)
[25] Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**(5), 506–521 (May 1999), preliminary version in *ICCAD '96*
[26] Martins, R., Manquinho, V.M., Lynce, I.: Open-WBO: A modular MaxSAT solver. In: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14). Lecture Notes in Computer Science, vol. 8561, pp. 438–445. Springer (Jul 2014)
[27] Mikša, M., Nordström, J.: Long proofs of (seemingly) simple formulas. In: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14). Lecture Notes in Computer Science, vol. 8561, pp. 121–137. Springer (Jul 2014)
[28] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference (DAC '01). pp. 530–535 (Jun 2001)
[29] Open-WBO: An open source version of the MaxSAT solver WBO. http://sat.inesc-id.pt/open-wbo/
[30] Pseudo-Boolean competition 2016. http://www.cril.univ-artois.fr/PB16/ (Jul 2016)
[31] Roussel, O., Manquinho, V.M.: Pseudo-Boolean and cardinality constraints. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 22, pp. 695–733. IOS Press (Feb 2009)
[32] Sat4j: The Boolean satisfaction and optimization library in Java. http://www.sat4j.org/
[33] Sheini, H.M., Sakallah, K.A.: Pueblo: A hybrid pseudo-Boolean SAT solver. Journal on Satisfiability, Boolean Modeling and Computation **2**(1-4), 165–189 (Mar 2006), preliminary version in *DATE '05*
[34] Spence, I.: sgen1: A generator of small but difficult satisfiability benchmarks. Journal of Experimental Algorithmics **15**, 1.2:1–1.2:15 (Mar 2010)
[35] Van Gelder, A., Spence, I.: Zero-one designs produce small hard SAT instances. In: Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10). Lecture Notes in Computer Science, vol. 6175, pp. 388–397. Springer (Jul 2010)
[36] Vinyals, M., Elffers, J., Giráldez-Cru, J., Gocht, S., Nordström, J.: In between resolution and cutting planes: A study of proof systems for pseudo-Boolean SAT solving (Jul 2018), to appear

[37] Wetzler, N., Heule, M., Hunt Jr., W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14). Lecture Notes in Computer Science, vol. 8561, pp. 422–429. Springer (Jul 2014)