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

Jan Elffers[1], Jesús Giráldez-Crú[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 results demonstrate severe shortcomings in state-of-the-art PB solving techniques. Despite the fact that our benchmarks have linear-size tree-like CP proofs, the solvers often perform quite badly even for very small instances. Our analysis is that this shows that solvers need to explore stronger methods of pseudo-Boolean reasoning within cutting planes.
We make an empirical observation from the competition data that many of the easy crafted instances are also infeasible over the rational numbers, or have small strong backdoors to PB instances without rational solutions. This raises the intriguing question whether the existence of such backdoors can be correlated with easiness/hardness. However, for some of our constructed benchmark families even rationally infeasible instances are completely beyond reach. This indicates that PB solvers need to get better not only at Boolean reasoning but even at linear programming.
Finally, we compare CP-based solvers with CDCL and MIP solvers. For those of our benchmarks where the natural CNF encodings admit efficient resolution proofs, we see that the CDCL-based solver Open-WBO is orders of magnitude faster than the CP-based solvers cdcl-cuttingplanes and Sat4j (though it seems very sensitive to the ordering of the input). And the MIP solver Gurobi beats all of these solvers across the board.
These experimental results point to several crucial challenges in the quest for more efficient pseudo-Boolean solvers, and we also believe that a further study of our benchmarks could shed important 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 [6]. In this paper we consider the special case of *linear*

pseudo-Boolean constraints $\sum_i a_i \ell_i \geq A$ encoded as integral linear combinations of literals. In the decision problem *pseudo-Boolean solving (PBS)* one asks whether a collection of 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. Problems in many different fields can be expressed as such optimization problems. Clearly, PBO can be reduced to PBS by iteratively computing solutions and adding constraints that the value of the objective function should be better than the latest solution. 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 [2, 3] and developed in different directions. One line of research focuses on inference methods based on cutting planes [15, 8, 10], including works by Chai and Kuehlman [7], Sheini and Sakallah [31], and Dixon et al. [11]. 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 [7, 31], and that sometimes a very competitive approach can be to simply translate pseudo-Boolean constraints to CNF and use a *conflict-driven clause learning (CDCL)* SAT solver [12]. 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, 21]. Needless to say, this brief historical overview is very far from complete—see, e.g., the excellent survey [29] for more details.

Current state-of-the-art pseudo-Boolean solvers building on the first line of work discussed above include Open-WBO [28, 25], which reduces the problem to CNF [18] and applies CDCL search, and Sat4j [30, 20], 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 [13], which, as the name suggests, also implements conflict-driven search in cutting planes.

1.1 Our Investigations and Conclusions

The survey [29] 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 sentiment—pseudo-Boolean (PB) solvers are based on an exponentially stronger method of reasoning than CDCL solvers [4, 24, 27] 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 [34] as well as other crafted benchmarks that have been 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. 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 tentative 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 necessarily claim to provide only final, irrefutable conclusions. 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 claim to show clearly in this work that an in-depth study of PB solver performance on such benchmarks can provide interesting insights. This is especially so since the possibility to scale the size of the instances allows us to draw conclusions about asymptotic behaviour rather than just observing isolated data points.

The need for stronger Boolean reasoning in PB solvers 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 cutting planes proofs, and in most cases 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 struggle hopelessly for small formulas with linear-size DPLL proofs!

The most plausible explanation for the poor performance is 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. However, in addition to this our results also show that there is ample room for improvement in PB proof search heuristics.

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 PB encodings of so-called *subset cardinality formulas* [32, 33, 26]. However, the seemingly equally simple *even colouring formulas* [23] 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 they 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 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 it hard—but any formula without rational solutions has short proofs that PB solvers can find in theory [34], so we can ask if they find them in practice.

The answer from our experiments is again a firm "no"—there are instances without even 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*[3] *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.[4]

The need for stronger LP reasoning in PB solvers The refined rational hypothesis just discussed cannot explain all our findings, however, especially since solvers cannot always count on getting helpful hints. For some benchmark families—in particular, encodings of easy instances of *dominating set* on hexagonal grids—the formulas are extremely challenging even when the corresponding linear program has no rational solutions. 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, i.e., without any Boolean reasoning, and with these extra constraints added the formulas become trivial also in practice [34]. However, the solvers do not find these constraints on their own, and instead get stuck exploring parts of the search space where even the LP is infeasible. This shows that PB solvers would need to strengthen not only their Boolan 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. But for instances that are inherently pseudo-Boolean in nature, but where resolution can nevertheless efficiently simulate pseudo-Boolean reasoning if given a natural CNF encoding, we see that most often cdcl-cuttingplanes and Sat4j are

[3] 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 $F|_\rho$ that is in $\mathcal{F}$.

[4] To give a concrete illustration of the extended rational hypothesis, we mention that it is consistent with pseudo-Boolean encodings of 2-CNF formulas being easy. It follows from [1] that for an unsatisfiable 2-CNF formula there is some variable x such that assigning to it any truth value causes unit propagation to the other truth value. But unit propagation works also over the reals for the pseudo-Boolean encoding of clauses, and hence x is a strong backdoor of size 1.

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 can be made to match that of Open-WBO, but if left to its own devices cdcl-cuttingplanes will deviate from this decision order. It thus seems that 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.

A final observation is that the MIP solver Gurobi [16] is consistently better than all the PB solvers studied for all of our combinatorial benchmarks. 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. Thus, Gurobi is in some sense playing on home turf here. 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 in this paper might seem depressing in that they are mostly bad news for state-of-the-art pseudo-Boolean solvers. However, we would rather view our work as pointing forward to some crucial challenges that need to be overcome in the quest for more efficient pseudo-Boolean solvers, which can fulfil the vision of [29] 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

In our experiments we used the versions of the pseudo-Boolean solvers cdcl-cuttingplanes [13] (*cdcl-cp* from now on), Open-WBO [28, 25], and Sat4j [30, 20] submitted to the Pseudo-Boolean Competition 2016, where they ranked as the top three solvers in the category DEC-SMALLINT-LIN ("no optimisation, small integers, linear constraints, SAT+UNSAT answers"). As described in more detail in Section 3, we ran these solvers on decision problems encoded as linear constraints with small integer coefficients. Since our benchmarks are inspired by proof complexity, where one studies the complexity of certifying unsatisfiability, we focus almost exclusively on UNSAT instances.

The version of Sat4j submitted to the competition internally runs two sub-solvers in parallel (Resolution and CuttingPlanes) and returns the answer of the first of them solving the problem. We will refer to this version as *Sat4j*. The stand-alone version Sat4j Resolution showed a worse performance in the competition: it solved 11% less instances. The version Sat4j CuttingPlanes was

not submitted to the competition, but since we are particularly interested in analysing cutting planes-based solvers we include it in our experiments (referring to it as *Sat4jCP* from now on). However, we only show the results for Sat4jCP when they differ to the ones for Sat4j (i.e., when in the latter solver the formula was decided by the Resolution subsolver).

For our mixed integer programming experiments we first ran some experiments using the GNU Linear Programming Kit GLPK [14], and then extended them using the more advanced solver Gurobi [16].

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

3 Description of the Benchmarks

We next describe the families of benchmarks used in our experiments. We reiterate that all instances we consider are very easy for the cutting planes proof system, and so we are measuring the quality of proof search in pseudo-Boolean solvers on instances where they should be able to perform very well.

The well-known *pigeonhole principle (PHP) formulas* claim 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 $x^\ell_{i,j}$ and also introduce variables y^ℓ for $\ell \in [k]$. We can allow pigeon i^* in the ℓth instance to "take the emergency exit" by changing the pigeon axiom to $y^\ell + \sum_{j\in[n]} x^\ell_{i^*,j} \geq 1$. However, a special constraint $\sum_{\ell=1}^{k} y^\ell \leq k-1$ enforces that at most $k-1$ emergency exits are taken. 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.

We generate *subset cardinality formulas* [32, 33, 26] from 0/1 $n \times n$ matrices $A = (a_{i,j})$ with 4 ones in every row and column, except that one particular 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, and introducing a variable $x_{i,j}$ for every $a_{i,j} = 1$, the subset cardinality formula over 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 all $i, j \in [n]$. We consider these formulas over randomly generated matrices and for "fixed bandwidth" matrices with a fixed patterns of ones shifted down the diagonal of the matrix (more precisely, we have $a_{i,j} = 1$ for all $j = \left(\left(i-2+2^\ell\right) \bmod n\right)+1$ with $\ell = 0, 1, 2, 3$, plus an extra $a_{1,n} = 1$ in the top right corner). In addition to this version with greater-or-equal constraints, we consider a more restrictive version with equality constraints $\sum_{j\in R_i} x_{i,j} = \lceil |R_i|/2 \rceil$ and $\sum_{i\in C_j} x_{i,j} = \lfloor |C_i|/2 \rfloor$.

Let $G = (V, E)$ be a connected graph with all vertices v having even degree $\deg(v)$. The *even colouring (EC) formula* [23] on G consists of constraints $\sum_{e\ni v} x_e = \deg(v)/2$ claiming the existence of a 2-colouring of the edges such that every vertex has the same number of incident edges of each colour. This formula is unsatisfiable if and only if $|E|$ is odd. We study these formulas for

two families of graphs. First, we consider rectangular grids with m rows and n columns, where every vertex has edges to its 4 neighbours horizontally and vertically at distance 1 (including edges wrapping around so that the graph forms a torus), and then split one edge by inserting a degree-2 vertex to get an odd total number of edges. Second, we generate random regular graphs of even degree $2d$, splitting an edge to get an odd number of edges if d is even.

The *vertex cover formula* with constraints $\sum_{v \in V} x_v \leq S$ and $x_u + x_v \geq 1$ for all $(u, v) \in E$ encodes that a graph G has a vertex cover of size S (i.e., a subset of vertices such that every edge is incident to some vertex in the subset). As in [34], we consider long, narrow rectangular (toroidal) grid graphs with m rows and n columns for $m = \mathrm{O}(1)$ even. The minimal vertex covers for such graphs have size $m\lceil n/2 \rceil$. For n odd, we generate three versions by varying the value of S. First we set $S = m\lceil n/2 \rceil - 1$, the largest value such that the formula is still unsatisfiable (version *hard*). Second we set $S = mn/2$ (version *easy*), which is more obviously unsatisfiable but still has a rational solution with value $\frac{1}{2}$ for all variables. Finally we get a version with no rational solutions by setting $S = m\lfloor n/2 \rfloor - 1$ (version *norat*), and we also consider this size of the cover when n is even (version *norat-even*); in the latter case S is the largest value that makes the formula unsatisfiable both for Boolean and rational values. To obtain minimally unsatisfiable formulas we use a second family of graphs where we remove all vertical edges, yielding a collection of disjoint cycles, and we consider vertex cover instances for these graphs for the same four cases as above.

The *dominating set formula* for a graph G consists of constraints $\sum_{v \in V} x_v \leq S$ and $x_v + \sum_{u \in N(v)} x_u \geq 1$ for all $v \in V$, saying that G has a dominating set of size S (i.e., a subset of vertices such that every vertex either belongs to the subset or is a neighbour of a vertex in the subset). We study these formulas for hexagonal grids as represented in [34] with m rows and n columns, where the hexagons have been squashed so that the grid looks like a brick wall. We keep one dimension fixed and vary the other one, but since squashed hexagonal grids are not rotationally symmetric we have both instances with fixed number of rows m and instances with fixed number of columns n.

Since hexagonal grids are 3-regular any dominating set must have size at least $\lceil |V|/4 \rceil = \lceil mn/4 \rceil$, and so we choose $S = \lfloor |V|/4 \rfloor$. When $4 \nmid mn$ the resulting instance is rationally unsatisfiable, but for mn divisible by 4 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 [34]. Thus, in contrast to the other families some dominating set instances are satisfiable.

Linearized pebbling formulas[5] are generated for a directed acyclic graph with a unique sink vertex. For a given arity d, we associate with every vertex v variables $v_1, \ldots, v_d$. Let $d' = d$ if d is odd and $d' = d - 1$ otherwise. The formula consists of the following constraints:

- For every source vertex v the constraint $2\sum_{i=1}^{d} v_i \geq d'$.

[5] Some instances from this formula family were submitted to the Pseudo-Boolean Competition 2016 under the name `sumineq` (sum inequalities).

– For every non-source vertex w with predecessors u and v the constraint $2\sum_{i=1}^{d} w_i \geq \sum_{i=1}^{d}(u_i + v_i)$.
– For the unique sink z the constraint $2\sum_{i=1}^{d} z_i \leq d'$.

By arguing bottom-up we can establish that these formulas are unsatisfiable.

4 Experimental Evaluation

In this section we describe some results of our experiments, selected to address the questions raised in the introductory section, and our conclusions. By necessity our discussion is far from exhaustive; the reader may find more detailed results in the additional material.

A nice feature of our benchmarks is that each family contains instances of "the same" formula, only scaled in size by varying a parameter. When analyzing the results we can therefore study the asymptotic behaviour of the solvers as this scaling parameter (and the instance size) increases. Our plots illustrate this asymptotic behaviour, plotting the value of the scaling parameter on the x-axis.[6]

4.1 Boolean Reasoning

We begin by arguing that solvers need to improve on pseudo-Boolean reasoning, in particular use stronger derivation rules than saturation and better proof search. We use even colouring formulas as an example. All unsatisfiable EC formulas can be refuted in cutting planes using just two applications of division, and the proof is tree-like, that is analogous to DPLL if we compared to CDCL.

We would expect pseudo-Boolean solvers to run very fast on these formulas for any graph, but this is not the case. In order to collect at least some data we limit our experiments to $m \times n$ grids with $m = \mathrm{O}(1)$ a small constant, which have short proofs in resolution, and to random regular graphs, which look likely to be exponentially hard for resolution.[7]

On grids, for m even the formulas are trivial for both solvers, but for m odd they are hard. Interestingly, for this last case Sat4j (Resolution) performs better than Sat4jCP, suggesting that pseudo-Boolean solvers do not find a better proof than CDCL-based solvers. It is also notable that changing the vertex order from column-major to row-major, even if it should not have any effect, makes cdcl-cp find an efficient proof. This shows that there is ample room to improve on search heuristics.

To explain this difference we note that even colouring formulas always have a rational solution where all variables are assigned value $\frac{1}{2}$, however grids with an even number of rows and columns have backdoors of size 1 (namely, either of

[6] Note, in particular, that this seems to be a better way of visualizing asymptotic behaviour than using so-called cactus plots.

[7] It seems like a lower bound should be possible to obtain by the techniques in [5], but it does not quite work "off the shelf" and we are not aware of anyone having investigated this question in detail.

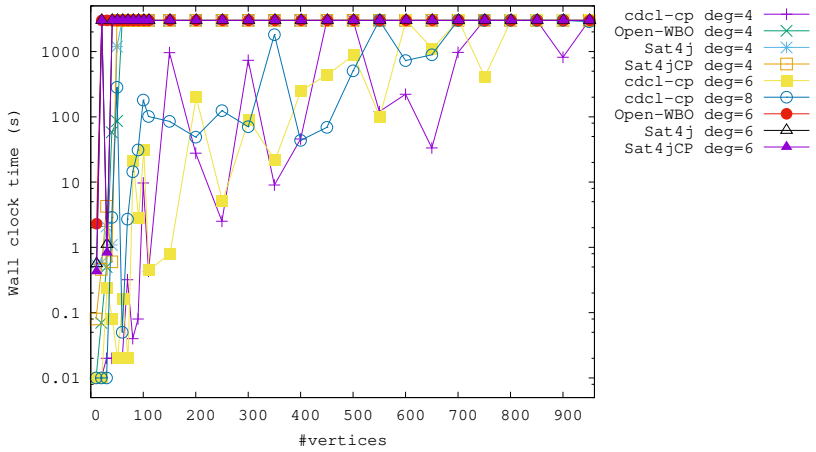

Fig. 1. Performance on even colouring formulas on random graphs

the edges incident to the degree-2 vertex), and as long as we keep the number of rows m even there is a backdoor of size at most $m = \mathrm{O}(1)$. If m is odd, however, the backdoor size jumps to $n - \mathrm{O}(1)$.

For random regular graphs EC formulas are hard for Open-WBO and Sat4j. For cdcl-cp these formulas are not easy, but it performs markedly better than the other two solvers, and it does not seem to be sensitive to the degree of the graph (see Figure 1). Since short proofs for these formulas rely on using the division rule, and cdcl-cp implements a limited version of division, this may be a reason for the difference in performance.

However, in the very few instances solved by Sat4j, the number of conflicts is not too unlike the number of conflicts of cdcl-cp, which raises the question of whether the difference in running time is due to implementation issues, as we discuss in Section 4.3. A more likely explanation is that Sat4j implements a very limited division rule, namely to divide by the gcd of the coefficients. It very rarely happens on EC formulas that this does the division we need. To be sure we would need proof logging, say an analogue to DRAT for CDCL, but there is no such thing for pseudo-Boolean solvers.

Another formula family where cdcl-cp performs better than Sat4j is linearized pebbling formulas. While they are easy for the resolution subsolver of Sat4j, they are extremely hard for Sat4jCP (see Figure 2). We observed that, with these formulas as input, Sat4j learns clauses with coefficients larger than 10^9 in a matter of seconds, while cdcl-cp rounds some clauses to keep coefficients small. Since the division rule is not needed to efficiently refute these formulas, it may be that having small coefficients is what helps cdcl-cp. Sat4j also uses large coefficients in instances of vertex cover and dominating set, where cdcl-cp performs better, but not in pigeonhole principle and subset cardinality, where both solvers are fast.

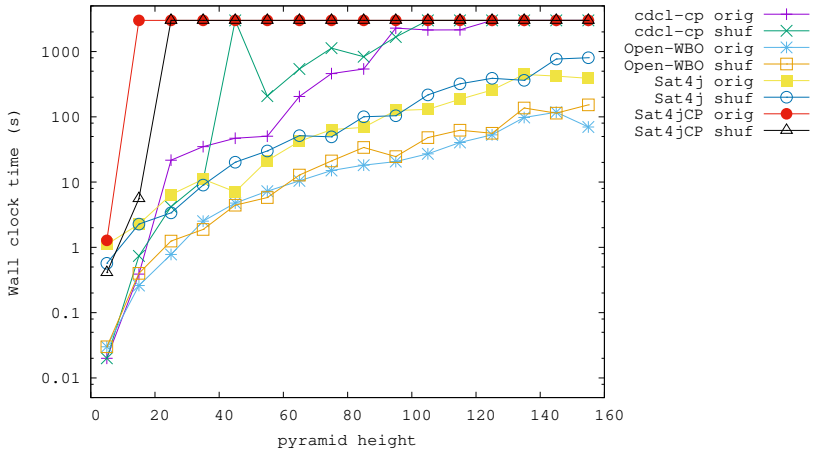

Fig. 2. Performance of cdcl-cp, Sat4j, Open-WBO, and Sat4jCP on linearized pebbling formulas on pyramids and arity 5, with and without randomly shuffling (*orig* and *shuf*, respectively).

To further study the role of backdoors we formulated the (extended) rational hypothesis. Let us next review what data tells us about it. PHP and subset cardinality formulas do not have rational solutions, and the fact that instances are trivial for both cdcl-cp and Sat4j supports the rational hypothesis. We considered PHP formulas with emergency exits as a way of generating potentially harder instances which still do not have rational solutions. However, this fails to fool cdcl-cp, and hence can be interpreted as circumstantial evidence in favour of the rational hypothesis for this solver (but not for Sat4j).

On even colouring we just mentioned that cdcl-cp often runs fast on when the backdoor size is small even with standard heuristics, or by slightly changing the encoding of the formula. This supports the hypothesis about a connection between hardness and size of backdoors to rational unsatisfiability. For random regular graphs it seems quite likely that there are no small backdoors, but we have not attempted to establish this by rigorous calculations.

The vertex cover instances "norat" and "norat-even" without rational solutions are easier than the "easy" version, which is in turn easier than the "hard" version. These results are consistent with the rational hypothesis since the size of the backdoor is 0 for the versions without rational solutions and 1 for the "easy" version (rational solutions disappear after branching on any vertex), whereas the smallest backdoor we found for the "hard" version has size $m - 1$ (rational solutions are eliminated after branching on any $m - 1$ vertices in the same column). This holds for both grids and collections of cycles.

Dominating set formulas on hexagonal grids have rational solutions when the size of the grid is divisible by 4. In this case, empirical evidence suggests that there are strong backdoors of size 3 (obtained by considering any vertex and two

of its neigbours). The best fixed variable ordering that we have found for cdcl-cp is to order nodes by columns and branch always with positive phase. However, this fixed order does not work well in all the cases; it does not seem to perform well when the number of rows is odd. We plan to investigate other potential good orders as a future work, but as of now our experimental results for these formulas do *not* support the rational hypothesis.

Linearized pebbling formulas have backdoors of size d (branching on the d variables $z_1, \ldots, z_d$ associated with the sink removes all rational solutions). We are not yet able to make cdcl-cp and Sat4jCP run fast on these formulas, hence they remain problematic instances for the rational hypothesis. We plan to investigate this further, and we believe that it should be possible to find heuristics for cdcl-cp and Sat4jCP to obtain a performance that is competitive with that of OpenWBO and Sat4j (Resolution).

4.2 Linear Programming

To exemplify the claim that PB solvers need to improve at linear programming, and not only at Boolean reasoning, we use experiments on dominating set formulas. As an overview, these formulas are very hard for Sat4j and also for cdcl-cp. They are manageable for Open-WBO when the fixed dimension is small but become very hard as this dimension grows. Also, Open-WBO is extremely sensitive to random shuffling. We believe that this is explained by that how well the CNF encoding of the cardinality constraint $\sum_{v \in V} x_v \leq S$ works is highly dependent of the ordering of the vertices, as we discuss for vertex cover in Section 4.3, and just as for vertex cover cdcl-cp is less sensitive to shuffling. We present some results in Figure 3.

Going to the point, it is possible to make the solvers run fast modifying the input formulas as follows. For each vertex v we have a greater-or-equal (GEQ) constraint stating that this vertex is dominated. Since the hexagonal grids are 3-regular, and since the required dominating set size is at most $|V|/4$, it follows that exactly one of the vertices in $\{v\} \cup N(v)$ is in the dominating set. These less-or-equal (LEQ) constraints can provably be derived using the subset of linear programming rules used by cdcl-cp and Sat4j [34], although so far we have not been able to get the solvers to realize this. However, if we add these LEQ constraints to the original formula, then it becomes trivial, as can be seen in Figure 3, hence stronger linear programming is the only missing part. This figure also illustrates that cdcl-cp with a good fixed order is somewhat competitive with Open-WBO.

4.3 PB vs CDCL and MIP

When comparing pseudo-Boolean solvers to CDCL solvers we get mixed results. On pigeonhole principle and subset cardinality formulas both pseudo-Boolean solvers perform very well, while Open-WBO provably cannot match them. On the remaining formulas Open-WBO is surprisingly competitive, but often brittle. Since pseudo-Boolean solvers do not perform well enough we had to focus on

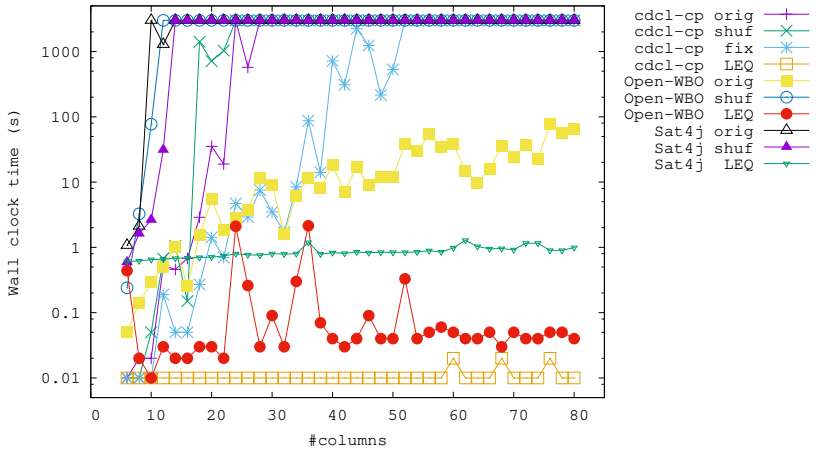

Fig. 3. Performance for dominating set on hexagonal grids with 7 rows with random shuffling (*shuf*) and without (*orig*) and also with added LEQ constraints as well as for cdcl-cp with fixed order (*fix*).

instances that are simple enough to have short resolution proofs, and Open-WBO is often good at finding them, but not resilient to formula perturbations. On the other hand, we can tweak cdcl-cp to match Open-WBO (although we expect it should do much better).

A good example of this phenomenon are vertex cover formulas. The performance of cdcl-cp and Sat4j on these formulas is quite poor in general, though cdcl-cp is clearly better than Sat4j. 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 implementation issues.

Open-WBO performs quite well for all almost all our families of vertex cover instances (except for the hard version on collections of cycles), which indicates that the encoding to CNF admits an efficient resolution proof. Since the edge constraints $x_u + x_v \geq 1$ are already disjunctive clauses, the performance is likely to depend on how the cardinality constraint $\sum_{v \in V} x_v \leq S$ is encoded into CNF. A key aspect here is that the vertices are listed consecutively (to be more precise: in column-major order, i.e., with vertices in the same column ordered consecutively, followed by the vertices in the next column). This means that as Open-WBO explores the neighbourhood of a vertex, the auxiliary variables in the encoding of the cardinality constraint are likely to propagate and help the solver do the counting. Since we have not had access to the source code of Open-WBO we have not been able to verify that this explanation is correct, but it is supported by the fact that the performance of Open-WBO degenerates dramatically when instances are shuffled, which is not the case for cdcl-cp. Some of the above conclusions are illustrated in Figure 4.

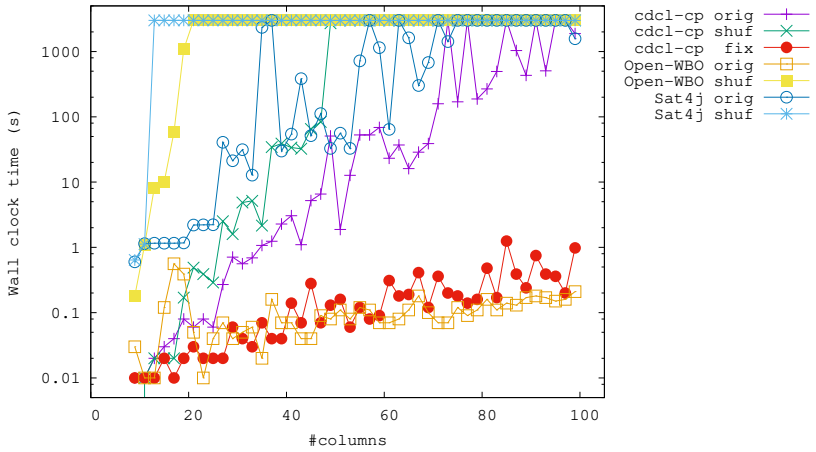

Fig. 4. Performance on vertex cover formulas on grids with 8 rows and vertex cover size $S \leq m\lfloor n/2 \rfloor - 1$ (version "norat") with shuffling (*shuf*) and without (*orig*), and for cdcl-cp also with fixed order (*fix*).

A further observation is that we can make cdcl-cp more efficient by forcing it to branch on the vertices in column-major order. The performance of this fixed order can be seen in Figure 4. However, the differences in terms of running time between Open-WBO and cdcl-cp with fixed order tend to increase for a large number of rows, with Open-WBO being clearly more efficient. Therefore, while these results remain consistent with the rational hypothesis, it would be interesting to see if cdcl-cp performance can be improved by using other fixed decision orders or fine-tuning other heuristics.

As an example of formulas that Sat4j and cdcl-cp can solve easily we have the pigeonhole principle. These have CP refutations in quadratic size that can be found with a linear number of conflicts. While both solvers only need a linear number of conflicts, and in fact they find exactly the same number of conflicts and probably also the same proof, a linear regression between the logarithms of the number of constraints and the running time shows that time grows like $\mathrm{O}\left(n^{3.2}\right)$ in cdcl-cp but like $\mathrm{O}\left(n^{5.0}\right)$ in Sat4j (see Figure 5). This helped us identify an implementation issue in Sat4j, and after fixing it runtimes became similar.

It is not surprising that these formulas are very hard for Open-WBO, since PHP is exponentially hard for resolution [17], and a careful study of the alternative proof of this lower bound in [5] reveals that it can be adapted to deal also with other common ways of encoding the at-most-1 constraints for the holes into CNF.

PHP formulas with emergency exits are always easy for cdcl-cp 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

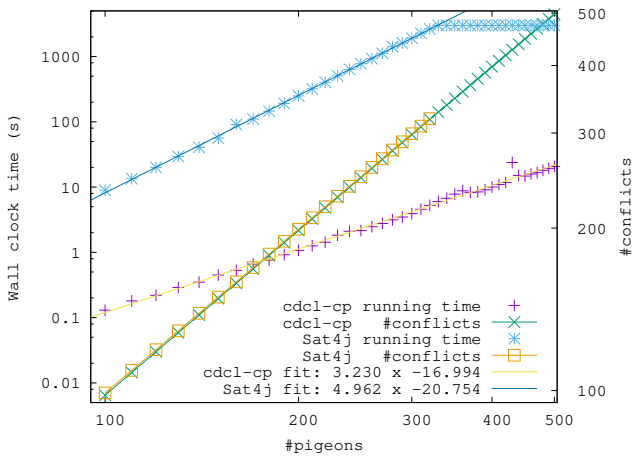

Fig. 5. Performance of cdcl-cp and Sat4j on PHP measured in terms of running time and #conflicts.

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-cp) solves these formulas efficiently, the number of conflicts seem to grow like $O(kn)$, but when Sat4j is not performing well the number of conflicts grows faster. Thus, in contrast to the results for standard PHP discussed above, here it seems that the quality of the actual proof search is worse in Sat4j.

Let us finally make some brief comments on the performance of MIP solvers. It is well-known that, in certain problems, MIP solvers can perform better than PB solvers. We analyzed the performance of Gurobi in our instances, and it does remarkably well—often much better, and never bad. This is not surprising, since our instances have tree-like proofs, and hence just branching and backtracking without learning is enough to solve them.

Since preprocessing can completely change the behaviour of a solver, we repeated the experiments with preprocessing turned off, but Gurobi still runs fast (other than some issues if equalities are expanded into two inequalities). This suggests that MIP search alone is very efficient for these instances, and that PB solvers could improve by incorporating techniques from their MIP counterparts.

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 essentially 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 proof search.

We perform extensive experiments on carefully constructed combinatorial benchmarks to investigate how efficiently these solvers implement their chosen methods of reasoning. Although all of our benchmarks 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 need to be strengthened, in particular, by exploiting the division rule.

For many of the instances we study. Sat4j and cdcl-cuttingplanes can be helped to run fast when given advice in the form of a good, fixed variable decision order. This is most often the case 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 when there are small backdoor sets such that any assignment to these backdoor variables eliminates all rational solutions. We find this to be a very intriguing connection. It seems highly interesting to investigate whether it can be the case more general 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 cutting planes-based solvers can be orders of magnitude slower. Interestingly, if cdcl-cuttingplanes is helped by being given a good variable order on such instances, then the performance is competitive with Open-WBO. However, when left to its own devices cdcl-cuttingplanes does not choose this order. This raises the question of whether the encoding to CNF that is used helps Open-WBO find a good variable order and stick with it. It should be noted, however, 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 general-purpose mixed integer programming solvers such as Gurobi. 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 is probably also due to that our benchmarks have been constructed to be easy for tree-like CP, and so they are amenable to branch-and-bound techniques.

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.

The pseudo-Boolean solver experiments for this paper were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at the High Performance Computing Center North (HPC2N) at Umeå University. Many of our benchmarks were generated using the tool `CNFgen` [9, 19], for which we gratefully acknowledge Massimo Lauria.

The 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.

References

[1] Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified Boolean formulas. Information Processing Letters 8(3), 121–123 (Mar 1979)

[2] 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*

[3] 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)

[4] 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)

[5] 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*

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

[7] 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*

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

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

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

[11] 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)

[12] Eén, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation 2(1-4), 1–26 (2006)
[13] Elffers, J.: cdcl-cuttingplanes: A conflict-driven pseudo-Boolean solver (2018), submitted to the *Pseudo-Boolean Competition 2016*; paper with solver description currently in review.
[14] GLPK (GNU linear programming kit). https://www.gnu.org/software/glpk/
[15] 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)
[16] Gurobi. http://www.gurobi.com/
[17] Haken, A.: The intractability of resolution. Theoretical Computer Science 39(2-3), 297–308 (Aug 1985)
[18] 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)
[19] 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)
[20] Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation 7, 59–64 (2010)
[21] 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*
[22] 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)
[23] Markström, K.: Locality and hard SAT-instances. Journal on Satisfiability, Boolean Modeling and Computation 2(1-4), 221–227 (2006)
[24] 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*
[25] 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)
[26] 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)
[27] 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)
[28] Open-WBO: An open source version of the MaxSAT solver WBO. http://sat.inesc-id.pt/open-wbo/
[29] 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)

[30] Sat4j: The Boolean satisfaction and optimization library in Java. http://www.sat4j.org/
[31] 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*
[32] Spence, I.: sgen1: A generator of small but difficult satisfiability benchmarks. Journal of Experimental Algorithmics 15, 1.2:1–1.2:15 (Mar 2010)
[33] 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)
[34] 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 (Feb 2018), submitted