Cutting to the Core of Pseudo-Boolean Optimization: Combining Core-Guided Search with Cutting Planes Reasoning

Jo Devriendt,[1,2,3] **Stephan Gocht,**[1,2] **Emir Demirović,**[4] **Jakob Nordström,**[2,1] **Peter J. Stuckey**[5]

[1] Lund University, Lund, Sweden
[2] University of Copenhagen, Copenhagen, Denmark
[3] KU Leuven, Leuven, Belgium
[4] TU Delft, Delft, The Netherlands
[5] Monash University, Melbourne, Australia

jo.devriendt@cs.lth.se, stephan.gocht@cs.lth.se, e.demirovic@tudelft.nl, jn@di.ku.dk, peter.stuckey@monash.edu

Abstract

Core-guided techniques have revolutionized Boolean satisfiability approaches to optimization problems (MaxSAT), but the process at the heart of these methods, strengthening bounds on solutions by repeatedly adding cardinality constraints, remains a bottleneck. Cardinality constraints require significant work to be re-encoded to SAT, and SAT solvers are notoriously weak at cardinality reasoning. In this work, we lift core-guided search to pseudo-Boolean (PB) solvers, which deal with more general PB optimization problems and operate natively with cardinality constraints. The cutting planes method used in such solvers allows us to derive stronger cardinality constraints, which yield better updates to solution bounds, and the increased efficiency of objective function reformulation also makes it feasible to switch repeatedly between lower-bounding and upper-bounding search. A thorough evaluation on applied and crafted benchmarks shows that our core-guided PB solver significantly improves on the state of the art in pseudo-Boolean optimization.

1 Introduction

The *Boolean satisfiability (SAT) problem* plays a fascinating dual role in computer science. Although it is an archetypal hard problem—proven *NP*-complete in (Cook 1971; Levin 1973) and widely believed to be exponentially hard in theory—at the same time it serves as the modelling language for the *conflict-driven clause learning (CDCL)* SAT solvers (Bayardo Jr. and Schrag 1997; Marques-Silva and Sakallah 1999; Moskewicz et al. 2001) that have emerged over the last two decades as highly practical tools for solving large-scale real-world problems in a wide range of application areas (Biere et al. 2021). This success has also led to exports of the conflict-driven paradigm beyond SAT solving to, e.g., *SAT-based optimization (MaxSAT)* (Fu and Malik 2006), *pseudo-Boolean (PB) optimization* (Chai and Kuehlmann 2005; Sheini and Sakallah 2006), *constraint programming (CP)* (Ohrimenko, Stuckey, and Codish 2009; Stuckey 2010), and *mixed integer programming (MIP)* (Achterberg 2007).

Our starting point in this work is the MaxSAT problem, which differs from SAT in that some of the constraints are declared to be soft, but with associated penalties for violating them, and where the goal is to minimize the total penalty of violated constraints. The *core-guided* approach introduced by (Fu and Malik 2006) optimistically assumes that this penalty is zero, and then tries to solve the resulting SAT problem under this assumption as described in (Eén and Sörensson 2003). If this attempt fails, the solver returns a *core* explaining why the assumption was too good to be true, and such cores are repeatedly used to update the estimate of the optimal solution and make new attempts with revised assumptions. Such techniques play a crucial role for the performance of modern MaxSAT solvers (Morgado et al. 2013), and have also been adapted to other paradigms such as *answer set programming (ASP)* (Andres et al. 2012) and constraint programming (Gange et al. 2020).

A technical barrier for efficient implementations of core-guided search, however, is that the process of using cores to strengthen bounds requires dealing with cardinality constraints. Such constraints are cumbersome to encode in the low-level language of propositional logic, and the *resolution* method on which CDCL SAT solvers are based (Beame, Kautz, and Sabharwal 2004) has severe limitations when it comes to cardinality reasoning (Haken 1985), affecting even core-guided approaches that use clauses to explain propagations by the cardinality constraints (Manquinho, Marques-Silva, and Planes 2009; Alviano, Dodaro, and Ricca 2015).

Our Contribution The simple but crucial observation underlying our work is that the MaxSAT problem of minimizing a weighted sum of penalties subject to constraints expressed in conjunctive normal form (CNF) is just a special case of pseudo-Boolean optimization (also known as 0-1 *integer linear programming*). An intriguing fact in this context is that there are PB solvers that borrow the conflict-driven paradigm from SAT but perform their reasoning using the *cutting planes* method (Cook, Coullard, and Turán 1987). Cardinality constraints are no problem for such solvers, since they operate with even more general PB constraints, and it is known that cutting planes applied to cardinality constraints is exponentially more powerful than resolution.

In view of this, it might seem like an attractive, and even obvious, proposition to combine pseudo-Boolean reasoning with core-guided search. In practice, however, harnessing

the theoretical power of the cutting planes method in PB solvers has turned out to be very challenging, to the extent that the best PB optimization solver NAPS (Sakai and Nabeshima 2015) in the most recent PB Competition in 2016 (`www.cril.univ-artois.fr/PB16/`) instead rewrites the input to CNF and runs a CDCL solver. One of the problems with PB solvers is that the increased degree of freedom make it hard to know how to best explore the search space, and for the same reason it is not a priori obvious what would be "the right way" of generalizing core-guided techniques to a PB setting.

In this paper, we report on our work on designing algorithms and heuristics for core-guided pseudo-Boolean solving. We implement different approaches in the state-of-the-art PB solver ROUNDINGSAT (Elffers and Nordström 2018), and perform an extensive evaluation on applied and crafted benchmarks from different domains.

The one-sentence summary of our results is that adding core-guided techniques dramatically improves the solver. Core-guided search with clausal cores, as in SAT, already enhances performance, but the cutting planes method also allows the solver to derive stronger, non-clausal, cores. These cores lead to better updates of the solution bounds, meaning that the solver can zoom in faster on the optimal solution. Even more strikingly, the fact that all cores and objective function reformulations can be expressed in the native format of the solver means that there is very little overhead. This makes it possible to go beyond *core-boosting* (Berg, Demirović, and Stuckey 2019), which combines an initial core-guided search phase with a longer upper-bounding linear search phase, and implement a fully *hybrid* mode that switches repeatedly back and forth between core-guided search and linear search at very little cost, similar to *interleaving* in ASP (Alviano et al. 2015). This hybrid mode is what gives the best performance overall.

We have also evaluated popular heuristics from core-guided MaxSAT solvers such as using *stratification* (Ansótegui et al. 2012) and *independent cores* (Berg and Järvisalo 2017) during core-guided search, and fixing the phase to that of the incumbent solution during linear search (Demirović, Chu, and Stuckey 2018; Demirović and Stuckey 2019) rather than using standard phase saving as in (Pipatsrisawat and Darwiche 2007). Here the results are not so clear-cut. How to set the phase does not seem to have a decisive influence. Stratification and independent cores have a much less positive impact than we expected—these settings are good for some classes of benchmarks, but for others they make performance notably worse (which is particularly pronounced for independent cores).

Overall, adding core-guided search to ROUNDINGSAT dramatically improves the solver, to the extent that it is now better by a wide margin than the latest versions of both NAPS (Sakai and Nabeshima 2015) and SAT4J (Le Berre and Parrain 2010) for the PB Competition 2016 benchmarks.

2 Preliminaries

We start with a review of the basics of pseudo-Boolean solving—this material is standard, and can be found, e.g., in (Buss and Nordström 2021). A *literal* ℓ over a Boolean variable x is x itself or its negation $\overline{x} = 1 - x$, where variables take values 0 (false) or 1 (true) and where we define $\overline{\overline{x}} = x$ for convenience. A *pseudo-Boolean (PB) constraint* C is a 0-1 integer linear inequality

$$\sum_i a_i \ell_i \geq B \ , \tag{1}$$

which without loss of generality we always assume to be in *normalized form*; i.e., all literals ℓ_i are over distinct variables and the coefficients a_i and the *degree (of falsity)* B are non-negative integers. A cardinality constraint is a PB constraint in normalized form where all coefficients are 1. We use equality $\sum_i a_i \ell_i = B$ as syntactic sugar for the pair of inequalities $\sum_i a_i \ell_i \geq B$ and $\sum_i -a_i \ell_i \geq -B$ (but rewritten in normalized form).

The *weakening* rule $\mathsf{weaken}(C, \ell_j) \equiv \sum_{i \neq j} a_i \ell_i \geq B - a_j$ removes a literal ℓ_j from the constraint by subtracting its coefficient from the right-hand side, and $\mathsf{weaken}(C, L)$ for a set of literals L performs this operation for all $\ell_j \in L$. The division rule $\mathsf{divide}(C, d) \equiv \sum \lceil a_i/d \rceil \ell_i \geq \lceil B/d \rceil$ divides all coefficients and the degree by $d \in \mathbb{N}^+$ and rounds up. The operation $\mathsf{round2card}(C)$ computes a cardinality constraint over the literals in C with the degree equal to the minimum number of literals that must be set to true in order to satisfy C.

A *PB formula* is a conjunction $F = \bigwedge_j C_j$ of PB constraints. Note that a *clause* $\ell_1 \vee \cdots \vee \ell_k$ is equivalent to the constraint $\ell_1 + \cdots + \ell_k \geq 1$, so formulas in *conjunctive normal form (CNF)* are special cases of PB formulas.

A *(partial) assignment* ρ is a (partial) function from literals to $\{0, 1\}$, where we write $\rho(x) = \rho(\overline{x}) = *$ if x is not in the domain of ρ and define $\rho(\overline{x}) = 1 - \rho(x)$ otherwise. If ρ is partial, then it is also referred to as a *restriction*, and the *restricted* constraint $C|_\rho$ is obtained by substituting values for all assigned variables and adjusting the degree appropriately, i.e.,

$$C|_\rho = \sum_{\rho(\ell_i)=*} a_i \ell_i \geq B - \sum_{\rho(\ell_i)=1} a_i \ . \tag{2}$$

For a PB formula $F = \bigwedge_j C_j$ we define $F|_\rho = \bigwedge_j C_j|_\rho$. The constraint C is *satisfied* by ρ if $\sum_{\rho(\ell_i)=1} a_i \geq B$, (or, equivalently, if the restricted constraint (2) has a non-positive degree and hence is trivial). A PB formula is satisfied by ρ if all constraints in it are, in which case it is *satisfiable* and ρ is a *solution*. If there is no satisfying assignment, the formula is *unsatisfiable*.

A constraint C is said to *unit propagate* the literal ℓ under ρ if $C|_\rho$ cannot be satisfied unless $\ell \mapsto 1$. During *unit propagation* on F under ρ, we extend ρ iteratively by any propagated literals $\ell \mapsto 1$ until an assignment ρ' is reached under which no constraint $C \in F$ is propagating, or under which some constraint C propagates a literal that has already been assigned to the opposite value. The latter scenario is referred to as a *conflict*, since ρ' *violates* the constraint C in this case, and ρ' is called a *conflicting* assignment.

A *pseudo-Boolean optimization* problem consists of a PB formula F and an objective function $O \equiv \sum_{i=1}^{m} a_i \ell_i + c_o$. We will abuse notation slightly and write $(a, \ell) \in O$ to obtain coefficient-literal pairs from the objective. Given an assignment ρ we write $\rho(O)$ to denote the value of the objective function under ρ. Without loss of generality we assume

that all coefficients in the objective are positive and that we want to minimize the objective function. An *optimal solution* is a satisfying assignment for F with minimum objective value $\rho(O)$ among all solutions.

Let $\text{Vars}(F)$ ($\text{Lits}(F)$) denote the variables (literals) appearing in F and analogously for O. Given a PB optimization problem, a *fresh variable* is a variable that does not appear in the formula or the objective function.

The idea of *linear search*, a widely used approach for PB optimization, is to find a solution ρ to the formula F, after which the constraint $O \leq \rho(O) - 1$ (in normalized form) is added to F. This can be repeated until F turns unsatisfiable, at which point the solution last found is the optimal solution.

Core-guided MaxSAT Solving *Maximum satisfiability (MaxSAT)* can be viewed as a PB optimization problem with a CNF formula, but many MaxSAT solvers use not only linear search but also *core-guided* approaches that work as follows (expressed in pseudo-Boolean notation).

Given an objective function O, we build a set of assumptions $A = \text{Lits}(O)$ and solve the formula $F \cup \{\overline{\ell_i} \mid \ell_i \in A\}$. There are two cases: either we find a solution (which must be optimal, since the objective value is zero under A) or the problem is unsatisfiable. In the latter case, the solver can be made to return a subset $\kappa \subseteq A$ of the assumption that force unsatisfiability. This subset κ, treated as a clause $\sum_{\ell \in \kappa} \ell \geq 1$, is called an *unsatisfiable core*, and is implied by F since one of the assumptions must be falsified in any solution. Core-guided methods then reformulate the problem to take this information into account, deducing that the objective value must be at least $a_{min} = \min_{\ell_i \in \kappa} a_i$. The *OLL method* (Andres et al. 2012; Morgado, Dodaro, and Marques-Silva 2014) introduces new Boolean variables z_j that represent the (lower bounds of the) sum $1 + \sum_j z_j = \sum_{\ell \in \kappa} \ell$, and essentially rewrites the objective to $O + a_{min}(1 + \sum_j z_j - \sum_{\ell \in \kappa} \ell)$. This is a new MaxSAT problem, for which we can repeat the procedure described above again. The whole process terminates when a solution is found, which is guaranteed to be optimal since it assigns zero to all literals in the rewritten objective function.

3 Overview of the Optimization Algorithm

The general idea of our PB optimization approach is shown in Algorithm 1, which uses an *incremental* PB solver. The interface to the solver is similar to that of an incremental SAT solver (Eén and Sörensson 2003) and has two methods, one for adding constraints and one for solving the problem. The solve method $solve(A)$ also takes a (potentially empty) set of literals A and returns either $sat(\rho)$ where ρ is a full assignment satisfying the added constraints and the assumptions A; or $unsat(C)$ where C is a PB constraint implied by the added constraints that is falsified by the assumptions. We call this constraint a *core*. Note that in contrast to the MaxSAT setting described in Section 2, such a core can now be an arbitrary, non-clausal PB constraint.

In each iteration, Algorithm 1 will refine either the lower or upper bound. If a solution is found a constraint is added that only allows strictly better solutions. If a core is returned

Algorithm 1 PB Optimization with Core Extraction.

```
1:  procedure OPTIMIZE(F, O)
2:      lb ← 0;   ub ← ∞;   O' ← O
3:      solver.add(F)
4:      while ub − lb > 0 do
5:          pick set of assumption literals A ⊆ Lits(O')
6:          result ← solver.solve({¬ℓ_i | ℓ_i ∈ A})
7:          if result ≡ sat(ρ) then
8:              ub ← ρ(O)
9:                      ▷ improves best solution by at least 1
10:             solver.add(O < ub)
11:         else            ▷ unsatisfiable under assumptions
12:             let result ≡ unsat(C)
13:             lb ← improveBound(O', C)
14:                     ▷ improves lower bound by at least 1
15:             E ← encoding for reformulation variables
16:             O' ← reformulate(O', E)
17:             solver.add(E)
18:     return ub
```

we will reformulate the objective function so that the best known lower bound is the constant part of the objective and so that all coefficients remain non-negative.

Example 1. To illustrate Algorithm 1, suppose we want to minimize $x_1 + x_2 + x_3 + x_4$ subject to $x_1 + x_2 + 2x_5 \geq 2$ and $x_3 + x_4 - 2x_5 \geq 0$. If we start by assuming all variables to false, the solver obtains the core $x_1 + x_2 + x_3 + x_4 \geq 2$ by adding the two constraints in the formula together. This core gives us a lower bound of 2. Next we introduce new variables z_1, z_2 that encode the value of the sum of the x_i variables by adding the constraint $z_1 + z_2 + 2 = x_1 + x_2 + x_3 + x_4$ to the solver. Note that this equality represents a reformulation $z_1 + z_2 + 2$ of the objective function, so we can now continue by minimizing this reformulated objective $z_1 + z_2 + 2$, which contains the just derived lower bound 2 as constant term. Suppose we perform the next iteration without assumptions and let us say the solver produces a solution with objective value 3. We add the constraint $x_1 + x_2 + x_3 + x_4 < 3$ and continue. Because the gap between best solution and lower bound is 1 the solver will terminate in the next iteration by finding an optimal solution. □

This example also demonstrates the advantages of using a PB solver. Firstly, the produced cores are not clauses but more general PB constraints, thanks to which we can obtain larger increases in the lower bound. Secondly, it is very easy to add the upper bound after a solution is found, because the upper bound is represented as a single constraint that the solver can handle natively. Finally, the PB solver can employ strong reasoning based on the cutting planes proof system.

To fully utilize the potential of combining PB optimization with core-guided search we need to overcome multiple challenges: How do we extract cores? How can we guarantee a sound reformulation for arbitrary PB cores? How do we avoid introducing huge numbers of variables during reformulation that would slow down the solver? How do we choose the set of assumptions? We will discuss these questions in what follows.

4 Contributions

Lifting Incremental Solving to PB Incremental PB solving is similar to incremental SAT solving, but there are more degrees of freedom to core extraction, which we explore in this section. Similar to SAT, we detect unsatisfiability with respect to the assumptions when a learned constraint L is generated that causes a conflict after propagating the assumptions. All literals in L which were falsified by propagation can be systematically eliminated from L to generate a pseudo-Boolean *core constraint* C.

One difference from MaxSAT is that a PB core constraint can still contain non-assumption literals. These can be safely eliminated by weakening. The resulting *decision literal core* is a PB constraint only involving assumption literals. Another difference is that it could be that a valid core constraint is encountered before all falsified non-assumption literals have been eliminated. If so, we can stop core extraction immediately and weaken all non-assumption literals to obtain a constraint that we will call an *early core*. Both of these scenarios arise fairly frequently.

Postprocessing the Core to a Cardinality Constraint The cores obtained from the solver will in general be arbitrary PB constraints. It turns out to simplify matters (as we will explain later) to round this constraint to a *cardinality core constraint.* In general, a PB constraint $C \equiv \sum a_i \ell_i \geq B$ can imply multiple cardinality constraints, so we need to choose how to round. This is another example of a question that does not arise in MaxSAT solving. We consider two options: (a) the cardinality constraint that maximizes the lower bound increase; and (b) the shortest implied clause.

Maximal lower bound increase Given a cardinality core $\sum_{\ell \in X} \ell \geq B$, let $a_\ell > 0$ be the coefficient of the literal ℓ in the objective function to be minimized. The increase in the lower bound caused by this core is $B \cdot \min_{\ell \in X} a_\ell$. To find the best such core, we weaken all literals with small coefficients and evaluate the resulting rounded cardinality constraint in terms of lower bound increase as described in Algorithm 2.

Algorithm 2 Computation of cardinality constraint implied by C with maximal lower bound increase.

```
1:  procedure MAXLOWERBOUNDINCREASE(C, O)
2:      lb+ ← 0
3:      repeat
4:          Σ_{ℓ∈X} ℓ ≥ B ← round2card(C)
5:          m ← min_{ℓ∈X} a_ℓ
6:          if B · m > lb+ then
7:              lb+ ← B × m
8:              R ← Σ_{ℓ∈X} ℓ ≥ B
9:          C ← weaken(C, {ℓ | (m, ℓ) ∈ O})
10:     until B ≤ 0
11:     return R
```

Example 2. Consider the core $C \equiv 3x_1 + 3x_2 + 2x_3 + x_4 + x_5 \geq 7$ with objective $7x_1 + 3x_2 + 9x_3 + 6x_4 + 7x_5$. Then $\mathsf{round2card}(C) = x_1 + x_2 + x_3 + x_4 + x_5 \geq 3$. We compute $m = 3$ and set $lb^+ = 9$. We then weaken C to obtain $3x_1 + 2x_3 + x_4 + x_5 \geq 4$, which can be rounded to $R \equiv x_1 + x_3 + x_4 + x_5 \geq 2$. We compute $m = 6$ and set $lb^+ = 12$ storing R as the current best. We then weaken C obtaining $3x_1 + 2x_3 + x_5 \geq 1$, which yields the cardinality constraint $x_1 + x_3 + x_5 \geq 1$. We compute $m = 7$ but lb^+ does not increase. We weaken C to obtain $2x_3 \geq -3$, at which point the loop is exited and R is returned. □

Minimal Size Clause We construct a minimal size clause from C by weakening literals with the smallest coefficients until the degree is no greater than any remaining coefficient, after which we can construct a clause by division with the greatest coefficient a_{max}.

Example 3. As in Example 2, consider the core $C \equiv 3x_1 + 3x_2 + 2x_3 + x_4 + x_5 \geq 7$. Then we can weaken x_5, x_4 and x_3 to obtain $3x_1 + 3x_2 \geq 3$, with the remaining coefficients at least the weakened degree. The resulting shortest clause is $\mathsf{divide}(3x_1 + 3x_2 \geq 3, 3)$ or $x_1 + x_2 \geq 1$. □

Objective Reformulation After we obtained a cardinality constraint $\sum_{\ell \in X} \ell \geq B$ from a core, we reformulate the objective function such that the new lower bound is reflected in the constant part of the objective. Broadly speaking, we follow the OLL approach (Andres et al. 2012; Morgado, Dodaro, and Marques-Silva 2014) in MaxSAT solving, but with the crucial difference that there is no re-encoding of cardinality constraints to clauses. Instead, we introduce fresh variables z_i and add to the solver constraint database *sum encoding* constraints

$$\sum_{\ell \in X} \ell = B + \sum_{i=B+1}^{|X|} z_i \tag{3}$$

representing the sum of the literals in the core in unary, as well as *ordering constraints* $z_i \geq z_{i+1}$ to enforce that z_i is true if and only if $\sum_{\ell \in X} \ell \geq i$ holds. Here it is important to observe that if we had a general PB constraint in (3), we could be forced to introduce an exponential number of variables. Using (3), we can reformulate the objective function as

$$\sum_{(a,\ell) \in O} a\ell + c_o \tag{4a}$$

$$= \sum_{(a,\ell) \in O} a\ell + c_o + m(\sum_{\ell \in X} \ell) - m(\sum_{\ell \in X} \ell) \tag{4b}$$

$$\overset{(3)}{=} \sum_{(a,\ell) \in O} a\ell + c_o + m(B + \sum_{i=B+1}^{|X|} z_i) - m(\sum_{\ell \in X} \ell)\ , \tag{4c}$$

where m is the smallest coefficient in the objective function of literals in X. In this way, we ensure that $\sum_{(a,\ell) \in O} a\ell - m(\sum_{\ell \in X} \ell)$ still only has non-negative coefficients. This in turn means that the constant term $c_o + mB$ in the reformulated objective function is the new lower bound. Note that it is always possible to rewrite the objective function in this way, because we only assume to false literals that appear in the current reformulated objective function. Furthermore, every reformulation strictly increases the lower bound, so we will will eventually reach an optimal lower bound.

Lazy Variable Encodings One problematic issue with the objective function reformulation as described above is that every new reformulation can introduce many fresh variables, and as the number of new variables increases this can slow down the solver in future incremental calls. To alleviate this, one can introduce the z_i variables *lazily*, i.e., only when they are needed. Observe that in view of the ordering constraints $z_i \geq z_{i+1}$ we know that setting z_k to false forces z_i to false for all $i > k$ as well. Hence, we do not need the variables z_i for $i > k$ when assuming z_k as false. (This observation was also made in the context of incremental re-encoding of cardinality constraints to clauses in (Martins et al. 2014).)

To obtain a lazy encoding we take the sum encoding (3) and remove variables in a safe manner. Assume we only want to introduce variables up to z_k. Then we can write the equality in (3) as two inequalities

$$\sum_{i=B+1}^{|X|} z_i \leq \sum_{\ell \in X} \ell - B \leq \sum_{i=B+1}^{|X|} z_i \; . \tag{5}$$

For the lower bound on the left we can just omit the variables $z_i, i > k$, because this will only make the lower bound smaller. For the upper bound on the right we compensate for the removed variables by increasing the coefficient of z_k. This leads to the *lazy sum encoding*

$$\sum_{i=B+1}^{k} z_i \leq \sum_{\ell \in X} \ell - B \leq \\ \leq \sum_{i=B+1}^{k-1} z_i + (|X| - k + 1) z_k \; . \tag{6}$$

If we also want to remove z_i for $i < k$, we can do so in the upper bound on the right by replacing them with 1, and in the lower bound on the left we increase the coefficient of z_k so that the correct bound is implied for $z_k = 1$. This results in the *lazy reified encoding*

$$(k - B) z_k \leq \sum_{\ell \in X} \ell - B \leq \\ \leq (k - B - 1) + (|X| - k + 1) z_k \; . \tag{7}$$

Note that if $z_k = 0$, this simplifies to $B \leq \sum_{\ell \in X} \ell \leq k - 1$, and if $z_k = 1$ to $k \leq \sum_{\ell \in X} \ell \leq |X|$ as desired.

When a core is found, we still use the sum encoding (3) to reformulate the objective. This is implemented by maintaining an implicit representation of the reformulated objective function, storing for each core the factor used for reformulation as well as the number of variables that have not yet been introduced due to laziness. Instead of adding constraints as in (3) to the solver database, we only add the lazy encoding for a single variable. Due to other cores this variable can disappear from the reformulated objective and at this point we add the next variable and the corresponding lazy encoding to the solver. In case of the lazy sum encoding the solver can delete constraints that were introduced for variables with smaller index.

Utilizing the Upper Bound A further improvement can be achieved if an upper bound $u \in \mathbb{N}$ is known for the literals in the core, i.e., $\sum_{\ell \in X} \ell \leq u$. Such an upper bound means that we do not need to introduce all z_i variables but only variables up to $i = \min(u, |X|)$.

Hybrid Search The simplest strategy for choosing the assumptions is to not set any assumptions at all, which results in pure *linear search*. If the set of assumptions is non-empty, we refer to this as *core-guided search*. Since adding upper and lower bound constraints can be achieved with very low overhead in a native pseudo-Boolean setting, we explore a *hybrid search* variant where we switch back and forth between running the solver with and without assumptions, trying to roughly balance the time spent on linear and core-guided search, respectively (measuring not running time, however, but different statistics such as number of literals investigated during unit propagation, in order make the solver deterministic for reproducibility purposes).

5 Experiments

We have implemented the core-guided techniques discussed in Section 4 in the pseudo-Boolean solver RoundingSat (Elffers and Nordström 2018) and we have evaluated our implementation on four benchmark sets (converted to the standard OPB format used for PB solvers as needed):

- PB16: OPT-SMALL-INT benchmarks from the most recent PB Competition in 2016.
- MIP: 0-1 integer linear programming optimization problems from MIPLIB.
- KNAP: Knapsack benchmarks from (Pisinger 2005).
- CRAFT: Some crafted combinatorial benchmarks.

For comparing against other PB solvers, the PB16 benchmarks are the main target. We also study MIP and KNAP because they are two quite challenging sets of benchmarks for PB solvers, as observed in (Devriendt, Gleixner, and Nordström 2021). Finally, the crafted benchmarks are inspired by (Elffers et al. 2018; Vinyals et al. 2018), but have been generated with larger parameters so as to be more challenging. This allows us to "stress-test" the solvers by exposing them to problems that provably require sophisticated reasoning.

As hardware we used AMD Opteron 6238 nodes having 6 cores and 16 GiB of memory running Ubuntu 16.04.7. Each run was executed as a single thread on a node (leaving 5 cores unused to avoid timing issues due to competition for memory resources) with a 5000 second time-out limit. Binary, source code and detailed experimental results are available online (Devriendt et al. 2020).

Contribution of Core-Guided PB Techniques In order to investigate the impact of the different core-guided techniques we have developed for PB solving, we started by running extensive experiments with a large number of different settings to identify a good base configuration. Guided by these experiments, we chose a configuration that will be referred to as HYBRID in what follows. It uses hybrid solving interleaving core-guided and linear search phases, chooses the cardinality core that yields the largest increase of the lower bound for the objective function, and reformulates the objective using the lazy reified encoding. We then investigated three technical novelties of PB core-guided search:

1. non-clausal cores (comparing with HYBRIDCLAUSAL deriving clausal cores instead of cardinality cores),

	PB16 (1600)	MIP (291)	KNAP (783)	CRAFT (985)
HYBRID	968	78	306	639
HYBRIDCLAUSAL	937	75	298	618
HYBRIDNONLAZY	936	70	186	607
HYBRIDCLNONL	917	67	203	612
ROUNDINGSAT	853	75	341	309
COREGUIDED	911	61	43	595
COREBOOSTED	959	80	344	580
SAT4J	773	61	373	105
NAPS	896	65	111	345
SCIP	1057	125	765	642

Table 1: Number of instances solved to optimality for state-of-the-art solvers and ROUNDINGSAT core-guided variants.

2. lazy reformulation of the objective function (comparing with non-lazy reformulation in HYBRIDNONLAZY),
3. hybrid optimization with repeated switches back and forth between core-guided search and linear search (compared to linear search as in standard ROUNDINGSAT, pure COREGUIDED, and COREBOOSTED approaches).

We want to stress that the language of PB inequalities gives native support for an efficient implementation of this kind of approaches, in contrast to CNF. The top seven configurations of Table 1 shows that all three new features listed above significantly improve PB solver performance.

In more detail, Figure 1 provides a scatter plot of the number of cores needed to prove optimality for the configuration HYBRID as compared to the version HYBRIDCLAUSAL with clausal cores, *except that to get as clear a comparison as possible the plots are for pure core-guided search* with these solvers—adding linear search does not change the conclusions, but just makes the plot more fuzzy. Clearly, in the non-clausal settings fewer cores are needed.

In Figure 2 we study the number of new variables introduced by HYBRID as compared to the version HYBRIDNONLAZY that eagerly introduces all new variables in one go when the objective is reformulated. For many instances the lazy approach introduces orders of magnitude less variables, and this effect is especially pronounced for knapsack instances. We studied the two different lazy encodings (lazy sum and lazy reified) but did not see any significant differences in performance between the two—what is important is to avoid non-lazy reformulation. It is worth noting that even the weakest core-guided configuration HYBRIDCLNONL with clausal cores and non-lazy reformulations is clearly better than the original ROUNDINGSAT solver on the PB16 benchmarks, so core-guided search in itself is powerful.

The theoretical benchmarks in CRAFT give us a possibility to peek inside the solver, as it were, by exposing it to formulas expressing different combinatorial principles and thus requiring different forms of sophisticated reasoning. It is striking that on these benchmarks we see the clearest gains from the core-guided techniques.

Overall, a clear message is that adding core-guided techniques provides a dramatic boost for PB solving. And even

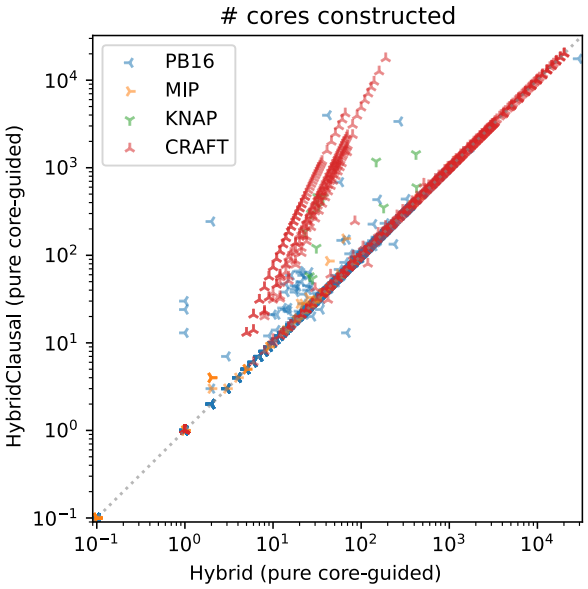

Figure 1: Number of cores during search for non-clausal (x-axis) versus clausal (y-axis) cores for instances solved by both approaches (but using pure core-guided optimization).

though the simplest version of core-guided search, without exploiting PB-specific techniques, can already provide major gains for some domains compared to the non-core-guided solver, our further PB optimizations help significantly to give more consistent performance improvements.

An interesting question is how to balance lower-bounding search using core-guided solving and upper-bounding linear search. As can be seen in Table 1, pure core-guided search (COREGUIDED) is not universally beneficial, and for KNAP even the hybrid mode is clearly not helpful compared to simple linear search. But it is interesting that our configuration COREBOOSTED with 10% *core-boosting* (Berg, Demirović, and Stuckey 2019) shows that a little bit of core-guided search can also help on these benchmarks. Overall, our new hybrid mode, switching repeatedly between core-guided and linear search, is the best. Importantly, this is not just an effect of hybrid providing a portfolio, as it were, of pure core-guided and pure linear search. For the crafted benchmarks, we verified that the hybrid solver even beats a parallel version where pure core-guided and pure linear search get to run side by side, each with a 5000 second time-out.

We have also evaluated the popular heuristics *stratification* (Ansótegui et al. 2012) and *independent cores* (Berg and Järvisalo 2017) from the core-guided MaxSAT literature. Figure 3 shows the effects of turning on stratification (HYBRIDSTRAT) and independent cores (HYBRIDSTRATIND) for the PB16 benchmarks. Looking at all benchmarks, switching on both stratification and independent cores helps for MIP and KNAP but does not change much for PB16 and is terrible for CRAFT. Stratification alone seems never to be a bad idea—we could have included it in our base configuration HYBRID and the con-

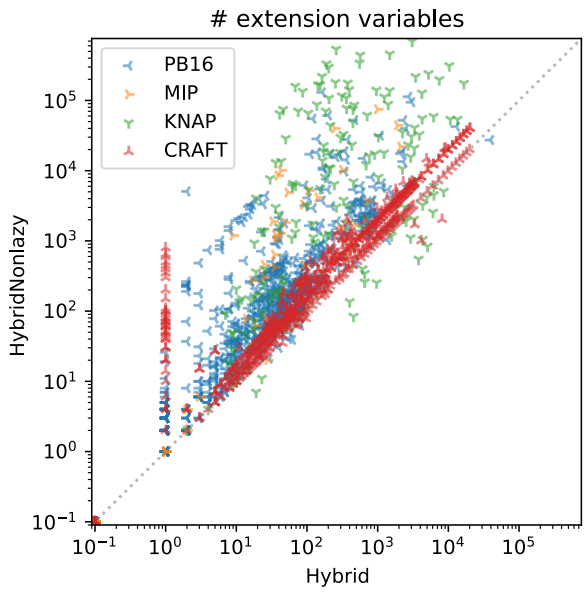

Figure 2: Number of new variables during search for lazy (x-axis) versus non-lazy (y-axis) objective reformulation for instances solved by both approaches.

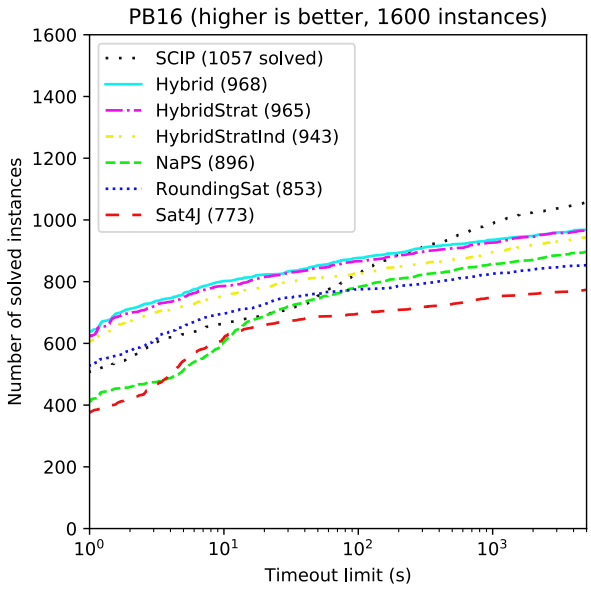

Figure 3: Cumulative plot for PB16 benchmarks.

clusions would not really have changed—but using independent cores can cause real problems for the wrong kind of benchmarks, which is especially clear for CRAFT. We are currently unable to explain why this is so.

Comparison to State of the Art In addition to comparing our core-guided PB solver to the original RoundingSat version, we evaluate two other PB solvers that performed well in the PB16 Competition as well as one MIP solver:

- Sat4J (Le Berre and Parrain 2010) commit c091d768. We use the *Both* strategy that essentially runs a CDCL solver and a cutting-planes-based PB solver in parallel.[1]
- NaPS (Sakai and Nabeshima 2015) commit 7aaa54f4. We use the *bignum* version as suggested to us by the authors. In contrast to RoundingSat and Sat4J, NaPS does *not* use cutting-planes-based reasoning but instead re-encodes the input to CNF and runs a CDCL solver.
- SCIP (Gamrath et al. 2020) version 7.0.0 using SoPlex version 5.0.0 as LP solver, with presolving support of Papilo 1.0 but without symmetry detection.

We present the results of this comparison in Table 1. Figure 3 gives a more detailed picture of the PB16 benchmarks.

Overall, our core-guided PB solver Hybrid decisively beats RoundingSat, Sat4J, and NaPS, with the notable exception that the dual-threaded version of Sat4J is best for KNAP benchmarks.

Sadly, PB solvers still struggle to compete with MIP solvers such as SCIP, and addressing this shortcoming seems to be the most interesting challenge for future research. One important factor to note is that presolving is a very important part of MIP performance, whereas current cutting-planes-based PB solvers essentially have no preprocessing. A natural approach would be to integrate the Papilo presolver with a cutting-planes-based PB solver and see what happens to performance. Another direction would be to combine core-guided solving with the use of linear programming relaxations, which is another core component of MIP solvers, and where the results in (Devriendt, Gleixner, and Nordström 2021) look promising. Already now, though, the results for some of the benchmarks in CRAFT show that there are problems that PB solvers solve very efficiently but that are beyond MIP solvers such as SCIP.

[1]This dual-threaded approach gets twice the CPU time of the other solvers, but we kept it so that our core-guided PB solvers would compete against the best version of Sat4J for each instance.

6 Concluding Remarks

In this work, we extend the resolution-based core-guided approach to pseudo-Boolean solvers using cutting planes reasoning, and show that this leads to dramatic improvements. The fact that PB solvers have native handling of PB constraints make them a very good fit for core-guided search.

Our work opens several directions for future work. One important problem is to find better strategies for balancing the lower- and upper-bounding phases in our hybrid approach for PB solvers. Independent core extraction is another technique that seems worth studying more closely—it has been successful for MaxSAT and CP solvers, but we see no such clear gains, and on the contrary this approach is largely detrimental for crafted instances. Finally, we would like to understand why MIP solvers are often (though not always!) much better than the best core-guided PB approach. Since preprocessing and LP relaxations are important components in MIP solvers, employing these techniques in PB core-guided search may be one key to further gains.

Acknowledgments

The first, second and fourth author were supported by the Swedish Research Council grant 2016-00782, and the fourth author also received funding from the Independent Research Fund Denmark grant 9040-00389B. Furthermore, the work was partially supported by the University of Melbourne AI Research group funding.

References

Achterberg, T. 2007. Conflict Analysis in Mixed Integer Programming. *Discrete Optimization* 4(1): 4–20.

Alviano, M.; Dodaro, C.; Marques-Silva, J. P.; and Ricca, F. 2015. Optimum Stable Model Search: Algorithms and Implementation. *Journal of Logic and Computation* 30(4): 863–897.

Alviano, M.; Dodaro, C.; and Ricca, F. 2015. A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI '15)*, 2677—2683.

Andres, B.; Kaufmann, B.; Matheis, O.; and Schaub, T. 2012. Unsatisfiability-Based Optimization in clasp. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP '12)*, volume 17 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 211–221.

Ansótegui, C.; Bonet, M. L.; Gabàs, J.; and Levy, J. 2012. Improving SAT-Based Weighted MaxSAT Solvers. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, 86–101. Springer.

Bayardo Jr., R. J.; and Schrag, R. 1997. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, 203–208.

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Towards Understanding and Harnessing the Potential of Clause Learning. *Journal of Artificial Intelligence Research* 22: 319–351. Preliminary version in *IJCAI '03*.

Berg, J.; Demirović, E.; and Stuckey, P. J. 2019. Core-Boosted Linear Search for Incomplete MaxSAT. In *Proceedings of the 16th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '19)*, volume 11494 of *Lecture Notes in Computer Science*, 39–56. Springer.

Berg, J.; and Järvisalo, M. 2017. Weight-Aware Core Extraction in SAT-Based MaxSAT Solving. In *Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming (CP '17)*, volume 10416 of *Lecture Notes in Computer Science*, 652–670. Springer.

Biere, A.; Heule, M. J. H.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition.

Buss, S. R.; and Nordström, J. 2021. Proof Complexity and SAT Solving. In (Biere et al. 2021), chapter 7, 233–350.

Chai, D.; and Kuehlmann, A. 2005. A Fast Pseudo-Boolean Constraint Solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24(3): 305–317. Preliminary version in *DAC '03*.

Cook, S. A. 1971. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, 151–158.

Cook, W.; Coullard, C. R.; and Turán, G. 1987. On the Complexity of Cutting-Plane Proofs. *Discrete Applied Mathematics* 18(1): 25–38.

Demirović, E.; Chu, G.; and Stuckey, P. J. 2018. Solution-Based Phase Saving for CP: A Value-Selection Heuristic to Simulate Local Search Behavior in Complete Solvers. In *International Conference on Principles and Practice of Constraint Programming*, 99–108. Springer.

Demirović, E.; and Stuckey, P. J. 2019. Techniques Inspired by Local Search for Incomplete MaxSAT and the Linear Algorithm: Varying Resolution and Solution-Guided Search. In *International Conference on Principles and Practice of Constraint Programming*, 177–194. Springer.

Devriendt, J.; Gleixner, A.; and Nordström, J. 2021. Learn to Relax: Integrating 0-1 Integer Linear Programming with Pseudo-Boolean Conflict-Driven Search. *Constraints* Preliminary version in *CPAIOR '20*.

Devriendt, J.; Gocht, S.; Demirović, E.; Nordström, J.; and Stuckey, P. J. 2020. Experimental Repository for "Cutting to the Core of Pseudo-Boolean Optimization: Combining Core- Guided Search with Cutting Planes Reasoning". URL https://doi.org/10.5281/zenodo.4043124.

Eén, N.; and Sörensson, N. 2003. Temporal Induction by Incremental SAT Solving. *Electronic Notes in Theoretical Computer Science* 89(4): 543–560.

Elffers, J.; Giráldez-Cru, J.; Nordström, J.; and Vinyals, M. 2018. Using Combinatorial Benchmarks to Probe the Reasoning Power of Pseudo-Boolean Solvers. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, 75–93. Springer.

Elffers, J.; and Nordström, J. 2018. Divide and Conquer: Towards Faster Pseudo-Boolean Solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, 1291–1299.

Fu, Z.; and Malik, S. 2006. On Solving the Partial MAX-SAT Problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT '06)*, volume 4121 of *Lecture Notes in Computer Science*, 252–265. Springer.

Gamrath, G.; Anderson, D.; Bestuzheva, K.; Chen, W.-K.; Eifler, L.; Gasse, M.; Gemander, P.; Gleixner, A.; Gottwald, L.; Halbig, K.; Hendel, G.; Hojny, C.; Koch, T.; Bodic, P. L.; Maher, S. J.; Matter, F.; Miltenberger, M.; Mühmer, E.; Müller, B.; Pfetsch, M.; Schlösser, F.; Serrano, F.; Shinano, Y.; Tawfik, C.; Vigerske, S.; Wegscheider, F.; Weninger, D.; and Witzig, J. 2020. The SCIP Optimization Suite 7.0. Technical report, Optimization Online. URL http://www.optimization-online.org/DB_HTML/2020/03/7705.html.

Gange, G.; Berg, J.; Demirović, E.; and Stuckey, P. J. 2020. Core-Guided and Core-Boosted Search for Constraint Programming. In *Proceedings of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '20)*, volume 12296 of *Lecture Notes in Computer Science*, 205–221. Springer.

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

Le Berre, D.; and Parrain, A. 2010. The Sat4j Library, Release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation* 7: 59–64.

Levin, L. A. 1973. Universal Sequential Search Problems. *Problemy peredachi informatsii* 9(3): 115–116. In Russian. Available at http://mi.mathnet.ru/ppi914.

Manquinho, V. M.; Marques-Silva, J. P.; and Planes, J. 2009. Algorithms for Weighted Boolean Optimization. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT '09)*, volume 5584 of *Lecture Notes in Computer Science*, 495–508. Springer.

Marques-Silva, J. P.; and Sakallah, K. A. 1999. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers* 48(5): 506–521. Preliminary version in *ICCAD '96*.

Martins, R.; Joshi, S.; Manquinho, V. M.; and Lynce, I. 2014. Incremental Cardinality Constraints for MaxSAT. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP '14)*, volume 8656 of *Lecture Notes in Computer Science*, 531–548. Springer.

Morgado, A.; Dodaro, C.; and Marques-Silva, J. P. 2014. Core-Guided MaxSAT with Soft Cardinality Constraints. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP '14)*, volume 8656 of *Lecture Notes in Computer Science*, 564–573. Springer.

Morgado, A.; Heras, F.; Liffiton, M.; Planes, J.; and Marques-Silva, J. 2013. Iterative and Core-Guided MaxSAT Solving: A Survey and Assessment. *Constraints* 18: 478–534.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, 530–535.

Ohrimenko, O.; Stuckey, P. J.; and Codish, M. 2009. Propagation via Lazy Clause Generation. *Constraints* 14(3): 357–391.

Pipatsrisawat, K.; and Darwiche, A. 2007. A Lightweight Component Caching Scheme for Satisfiability Solvers. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '07)*, volume 4501 of *Lecture Notes in Computer Science*, 294–299. Springer.

Pisinger, D. 2005. Where are the hard knapsack problems? *Computers & Operations Research* 32(9): 2271–2284.

Sakai, M.; and Nabeshima, H. 2015. Construction of an ROBDD for a PB-Constraint in Band Form and Related Techniques for PB-Solvers. *IEICE Transactions on Information and Systems* 98-D(6): 1121–1127.

Sheini, H. M.; and Sakallah, K. A. 2006. Pueblo: A Hybrid Pseudo-Boolean SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation* 2(1-4): 165–189. Preliminary version in *DATE '05*.

Stuckey, P. J. 2010. Lazy Clause Generation: Combining the Power of SAT and CP (and MIP?) Solving. In *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR '10)*, volume 6140 of *Lecture Notes in Computer Science*, 5–9. Springer.

Vinyals, M.; Elffers, J.; Giráldez-Cru, J.; Gocht, S.; and Nordström, J. 2018. In Between Resolution and Cutting Planes: A Study of Proof Systems for Pseudo-Boolean SAT Solving. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, 292–310. Springer.