Abstract Cores in Implicit Hitting Set MaxSat Solving

Jeremias Berg[1][0000−0001−7660−8061], Fahiem Bacchus[2], and Alex Poole[2]

[1] University of Helsinki, HIIT, Department of Computer Science, Finland
[2] University of Toronto, Department of Computer Science, Canada,
jeremias.berg@helsinki.fi,fbacchus@cs.toronto.edu

Abstract. Maximum Satisfiability (MaxSat) solving is an active area of research motivated by numerous successful applications to solving NP-hard combinatorial optimization problems. One of the most successful approaches to solving MaxSat instances arising from real world applications is the Implicit Hitting Set (IHS) approach. IHS solvers are complete MaxSat solvers that harness the strengths of both Boolean Satisfiability (SAT) and Integer Linear Programming (IP) solvers by decoupling core-extraction and optimization. While such solvers show state-of-the-art performance on many instances, it is known that there exist MaxSat instances on which IHS solvers need to extract an exponential number of cores before terminating. Motivated by the structure of the simplest of these problematic instances, we propose a technique we call abstract cores that provides a compact representation for a potentially exponential number of regular cores. We demonstrate how to incorporate abstract core reasoning into the IHS algorithm and report on an empirical evaluation demonstrating that including abstract cores into a state-of-the-art IHS solver improves its performance enough to surpass the best performing solvers of the most recent 2019 MaxSat Evaluation.

Keywords: Combinatorial Optimization · Maximum Satisfiability · MaxSat· Implicit Hitting Set · IHS

1 Introduction

Maximum Satisfiability (MaxSat), the optimisation extension of the Boolean Satisfiability (SAT) problem, has in recent years matured into a competitive and thriving constraint optimisation paradigm with several successful applications in a variety of domains [11,31,8,7,18,16,19]. As a consequence, the development of MaxSat solvers is an active area of research with the state-of-the-art solvers evaluated annually in the MaxSat Evaluations [5,4].

In this work, we focus on improving the Implicit Hitting Set (IHS) approach to complete MaxSat solving [4,14,29]. As witnessed by the results of the annual evaluations, IHS solvers are, together with core-guided [2,24,25,26,20] and model improving [22] algorithms, one of the most successful approaches to solving MaxSat instances encountered in practical applications. IHS solvers decouple MaxSat solving into separate *core extraction* and *optimisation* steps. By using a Boolean Satisfiability (SAT) solver for core extraction and an Integer Linear Programming (IP) optimizer, the IHS approach is able to exploit the disparate strengths of these different technologies.

Through this separation IHS solvers avoid increasing the complexity of the underlying SAT instance by deferring all numerical reasoning to the optimizer [13]. One drawback of the approach, however, is that on some problems an exponential number of cores need to be extracted by the SAT solver and given to the optimizer. In this paper we identify a seemingly common pattern that appears in the simplest problems exhibiting this exponential worse case. We propose a technique, which we call *abstract cores*, for addressing problems with this pattern. Abstract cores provide a compact representation for a potentially exponential number of ordinary cores. Hence, by extracting abstract cores and giving them to the optimizer we can in principle achieve an exponential reduction in the number of constraints the SAT solver has to extract and supply to the optimizer. The net effect can be significant performance improvements.

In the rest of the paper we formalize the concept of abstract cores and explain how to incorporate them into the IHS algorithm both in theory and practice. Finally, we demonstrate empirically that adding abstract cores to a state-of-the-art IHS solver improves its performance enough to surpass the best performing solvers of the 2019 MaxSat evaluation.

2 Preliminaries

MaxSat problems are expressed as CNF formulas $\mathcal{F}$ with weight annotations. A CNF formula consists of a conjunction ($\wedge$) of clauses, each of which is a disjunction ($\vee$) of literals, each of which is either a variable v of $\mathcal{F}$ (a positive literal) or its negation $\neg v$ (a negative literal). We will often regard $\mathcal{F}$ and clauses C as being sets of clauses and literals respectively. Hence, we write, e.g., $l \in C$, to indicate that literal l is in the clause C using set notation, and $C = (x, \neg y, z)$ to denote the clause $(x \vee \neg y \vee z)$.

A truth assignment τ maps Boolean variables to 1 (TRUE) or 0 (FALSE). It is extended to assign 1 or 0 to literals, clauses and formulas in the following standard way: $\tau(\neg l) = 1 - \tau(l)$, $\tau(C) = \max\{\tau(l) \,|\, l \in C\}$, and $\tau(\mathcal{F}) = \min\{\tau(C) \,|\, C \in \mathcal{F}\}$, for literals l, clauses C, and CNF formulas $\mathcal{F}$, respectively. We say that τ satisfies a clause C (formula $\mathcal{F}$) if $\tau(C) = 1$ ($\tau(\mathcal{F}) = 1$), and that the formula $\mathcal{F}$ is satisfiable if there exists a truth assignment τ such that $\tau(\mathcal{F}) = 1$.

A MaxSat instance $\mathcal{I} = (\mathcal{F}, wt)$ is a CNF formula $\mathcal{F}$ along with a weight function that maps every clause $C \in \mathcal{F}$ to a integer weight $wt(C) > 0$. Clauses C whose weight is infinite $wt(C) = \infty$ are called *hard clauses* while those with a finite weight are called *soft clauses*. $\mathcal{I}$ is said to be unweighted if all soft clauses have weight 1. We denote the set of hard and soft clauses of $\mathcal{F}$ by $\mathcal{F}_H$ and $\mathcal{F}_S$, respectively.

An assignment τ is a *solution* to $\mathcal{I}$ if it satisfies $\mathcal{F}_H$ ($\tau(\mathcal{F}_H) = 1$). The cost of a solution τ, $cost(\mathcal{I}, \tau)$, is the sum of the weights of the soft clauses it falsifies, i.e., $cost(\mathcal{I}, \tau) = \sum_{C \in \mathcal{F}_S} (1 - \tau(C)) \cdot wt(C)$. When the instance is clear from context we shorten notation to $cost(\tau)$. A solution τ is *optimal* if it has minimum cost among all solutions: i.e. if $cost(\tau) \leq cost(\tau')$ holds for all solutions τ'. The task in MaxSat solving is to find an (any) optimal solution. We will assume that at least one solution exists, i.e., that $\mathcal{F}_H$ is satisfiable.

To simplify our notation it will be useful to transform all of the soft clauses in $\mathcal{F}$ so that they become unit clauses containing a single negative literal. If $C \in \mathcal{F}_S$ is not in

the right form we replace it by the soft clause $(\neg b)$ and the hard clause $(C \vee b)$, where b is a brand new variable and $wt((\neg b)) = wt(C)$. This transformation preserves the set of solutions and their costs. We call the variables in the resulting set of unit soft clauses *blocking variables* or *b-variables* for short. Note that assigning a b-variable b the value TRUE is equivalent to falsifying its corresponding soft clause $(\neg b)$. We denote the set of b-variables of the transformed formula by $\mathcal{F}_B$, and write $wt(b)$ for a b-variable b to denote the weight of its underlying soft clause $wt(\neg b)$. With this convention we can write the cost of a solution τ more simply as $cost(\tau) = \sum_{b \in \mathcal{F}_B} wt(b) \cdot \tau(b)$. For any set B of b-variables we write $cost(B)$ to denote the sum of their weights.

In the MaxSat context a *core* κ is defined to be a set of soft clauses $\kappa \subseteq \mathcal{F}_S$ that are unsatisfiable given the hard clauses, i.e.. $\kappa \cup \mathcal{F}_H$ is unsatisfiable. This means that every solution τ, which by definition must satisfy $\mathcal{F}_H$, must falsify at least one soft clause in κ. Given that the soft clauses are of the form $(\neg b)$ for some b-variable b we can express every core as a clause $\kappa = \bigvee_{b \in \kappa} b$ containing only positive b-variables: one of these variables must be true. This clause is entailed by $\mathcal{F}_H$. We can also express κ as a linear inequality $\sum_{\{b|(\neg b) \in \kappa\}} b \geq 1$ that is also entailed by $\mathcal{F}_H$. A MaxSat *correction set* hs is dually defined to be a set of soft clauses $hs \subseteq \mathcal{F}_S$ whose removal renders the remaining soft clauses satisfiable with the hard clauses, i.e., $(\mathcal{F}_S - hs) \cup \mathcal{F}_H$ is satisfiable.

3 Implicit Hitting Set Based MaxSat Solving

Algorithm 1 shows the implicit hitting set (IHS) approach to MaxSat solving. Our specification generalizes the original specification of [13]. In particular, we use upper and lower bounds, terminating when these bounds meet, rather than waiting until the optimizer returns a correction set as in [13]. We use this reformulation as it makes it easier to understand our extension to abstract cores.

Starting from a lower bound of zero, an upper bound of infinity, and an empty set of cores $\mathcal{C}$ (line 3), the algorithm computes a minimum cost hitting set of its current set of cores $\mathcal{C}$. This is accomplished by expressing each core in $\mathcal{C}$ as its equivalent linear inequality $\sum_{b \in \kappa} b \geq 1$ and using the optimizer to find a solution hs with the smallest weight of true b variables (Figure 1a). This corresponds to computing the minimum weight of soft clauses that need to be falsified in order to satisfy the constraints imposed by cores found so far. Hence, $cost(hs)$ must be a lower bound on the cost of any optimal solution: every solution must satisfy these constraints. This allows us to update the lower bound (line 6) and exit the while loop if the lower bound now meets the upper bound. (Note that since new cores are continually added to the optimizer's model the lower bound will never decrease).

The optimizer's solution is then used to extract more cores that can be added to the optimizer's constrains for the next iteration. Core extraction is done by the `ex-cores` procedure shown in Algorithm 2. `ex-cores` extracts cores until it finds a solution τ. If the solution has lower cost than any previous solution the upper bound UB will be updated and this best solution stored in τ_{best}. The set of cores K extracted are returned and added to the optimizer's model potentially increasing the lower bound.

The original IHS formulation [13] extracted only one core from each optimizer solution, but this was shown to be a significant detriment to performance [15] requiring

```
1  Basic-IHS (F, wt)
       Input: A MaxSat instance (F, wt)
       Output: An optimal solution τ
2      LB ← 0; UB ← ∞
3      τ_best ← ∅; C ← ∅
4      while (TRUE) do
5          hs ← Min-Hs(F_B, C)
6          LB = cost(hs)
7          if (LB = UB) break
8          K ← ex-cores (hs, UB, τ_best)
9          if (LB = UB) break
10         C ← C ∪ K
11     return τ_best
```

Algorithm 1: IHS MaxSat

Min-Hs $(\mathcal{F}_B, \mathcal{C})$:

minimize: $\sum_{b \in \mathcal{F}_B} wt(b) \cdot b$

subject to:

$\sum_{b \in \kappa} b \geq 1 \qquad \forall \kappa \in \mathcal{C}$

$b \in \{0, 1\} \qquad \forall b \in \mathcal{F}_B$

return:

$\{b \mid b$ set to 1 in opt. soln$\}$

(a) IP for optimizing with cores

Fig. 1: IHS approach to MaxSat solving

too many calls to the optimizer. The procedure `ex-cores` gives one simple way of extracting more than one core from the optimizer's solution hs. It can be extended in a variety ways to allow extracting large numbers (hundreds) of cores from each optimizer solution [15,12,28]. In our implementation we used such techniques.

`ex-cores` (Alg. 2) uses a SAT solver and its assumption mechanism to extract cores. It first initializes the assumptions to force the SAT solver to satisfy every soft clause not in hs. More specifically, for every soft clause $(\neg b)$ not in hs, $\neg b$ is assumed, forcing the solver to satisfy this soft clause. Then it invokes `ex-cores-sub` which iteratively calls the SAT solver to find a solution satisfying $\mathcal{F}_H$ along with the current set of assumptions. After each core is found its b-variables are removed from the assumptions (line 11) so that on each iteration we require the SAT solver to satisfy fewer soft clauses. Since $\mathcal{F}_H$ is satisfiable, eventually the SAT solver will be asked to satisfy so few soft clauses that it will find a solution τ terminating the loop.

In the original IHS specification [13] IHS terminates with an optimal solution when the optimal hitting set hs is a correction set. This condition will also cause termination in our specification. In particular, before calling `ex-cores` the lower bound LB is set to $cost(hs)$ (Alg. 1, line 6). If hs is a correction set, a solution τ will be found by the SAT solver in the first iteration of Alg. 2, (line 7). That τ will have $cost(\tau) = cost(hs)$ as it cannot falsify any soft clause not in hs and cannot have cost less than the lower bound. Hence, on `ex-cores`'s return Algorithm 1 will terminate with $UB = LB$. As shown in [13] the optimizer's must eventually return a correction set. This means that the original proof that IHS terminates, returning an optimal solution given in [13] continues to apply our reformulated Algorithm 1.

Algorithm 1 can also terminate before the optimizer returns a correction set. In particular, τ_{best} can be set to an optimal solution (Alg. 2, line 9) well before we can verify its optimality. In this case termination can occur as soon as the optimizer has been given a sufficient number of cores to drive its lower bound up to $cost(\tau_{best})$, even if the optimizer's solution is not a correction set. In fact, termination in the IHS approach always requires that the optimizer be given enough constraints to drive the cost of its optimal solution up to the cost of the MaxSat optimal solution.

```
1  ex-cores (hs, UB, τ_best)
2  |  assumps = {¬b | b ∈ (F_B − hs)}
3  |  return ex-cores-sub(assumps, UB, τ_best)

4  ex-cores-sub (UB, τ_best)
5  |  K ← {}
6  |  while TRUE do
7  |  |  (sat?, κ, τ) ← sat-assume(F_H, assumps)
8  |  |  if (sat?) then
9  |  |  |  if (cost(τ) < UB) then τ_best ← τ; UB ← cost(τ)
10 |  |  |  return K
11 |  |  else K ← K ∪ {κ}; assumps ← assumps − {¬b|b ∈ κ}
```

Algorithm 2: Extracting multiple cores from a single optimizer solution

Example 1. With $\mathcal{F}_H = \{(b_1, b_2), (b_2, b_3), (b_3, b_4)\}$ and $\mathcal{F}_S = \{(\neg b_1), (\neg b_2), (\neg b_3), (\neg b_4)\}$ all having weight 1, Algorithm 1 will first obtain $hs = \emptyset$ from `Min-Hs` as there initially are no cores to hit. `ex-cores` will then SAT solve $\mathcal{F}_H$ under the assumptions $\neg b_1, \neg b_2, \neg b_3, \neg b_4$ trying to satisfy all softs not in hs. This is *unsat* and any of a number of different cores could be returned. Say that the core (b_1, b_2) is returned. `ex-cores` then attempts another SAT solve, this time with the assumptions $\neg b_3$ and $\neg b_4$. Now the SAT solver returns the core (b_3, b_4). Finally, the SAT solver will be called to solve $\mathcal{F}_H$ under the empty set of assumptions. Say that the solver finds the satisfying assignment $\tau = \{\neg b_1, b_2, \neg b_3, b_4\}$ setting UB to 2 and τ_{best} to τ. After returning to the main IHS routine, `Min-Hs` will be asked to compute an optimal solution to the set of cores $\{(b_1, b_2), (b_3, b_4)\}$. It might return $hs = \{b_1, b_4\}$ and set $LB = cost(hs) = 2$. Now LB is equal to UB and τ_{best} can be returned since it is an optimal solution. Note that in this example the solution returned by the optimizer, hs, is not a correction set.

As mentioned above IHS cannot terminate until its optimizer has been given enough constraints to drive the cost of an optimal solution up to be equal to the cost of an optimal MaxSat solution. As shown in [12] in the worst case this can require giving the optimizer an exponential number of constraints.

Example 2. Let n and r be integers with $0 < r < n$. Consider the MaxSat instance $\mathcal{F}^{n,r}$ with ${\mathcal{F}_H}^{n,r} = \text{CNF}(\sum_{i=1}^{n} b_i \geq r)$ and ${\mathcal{F}_S}^{n,r} = \{(\neg b_i) \mid 1 \leq i \leq n\}$, where $\text{CNF}(\sum_{i=1}^{n} b_i \geq r)$ is a CNF encoding of the cardinality constraint stating that at least r soft clauses must be falsified. The cost of every optimal solution is thus r; the maximum number of soft clauses that can be satisfied is $n - r$; and every subset of $n - r + 1$ soft clauses must be a core. Let $\mathcal{C}$ be the set of all of such cores. From the results of [12] we have that if the optimizer is given all cores in $\mathcal{C}$ it would yield solutions hs with $cost(hs) = r$; furthermore, if even one core of $\mathcal{C}$ is missing from the optimizer the optimizer solutions hs would have $cost(hs) < r$. This means that Algorithm 1 will have to extract $\binom{n}{n-r+1}$ cores for the optimizer before it can reach the cost of an optimal MaxSat solution and terminate. When r is close to $n/2$ the number of cores required for termination is exponential in n.

The results of the 2019 MaxSat Evaluation [4,5] witness this drawback in practice. The drmx-atmostk set of instances in the evaluation contain 11 instances with the same underlying structure as Example 2. Out of these, the IHS solver MaxHS [14,13], failed

to solve 8 out of 11 when given an hour for each instance, while the best performing solvers were able to solve all 11 instances in under 10s.

4 Abstract Cores

Example 2 shows that a significant bottleneck for the IHS approach on some instances is the large number of cores that have to be given to the optimizer. Thus, a natural question to ask is whether or not there exists a more compact representation of this large number of cores that can still be efficiently reasoned with by the IHS algorithm. In this section we propose *abstract cores* as one such representation. As we will demonstrate, each abstract core compactly represents a large number of regular cores. By extracting abstract cores with the SAT solver and then giving them to the optimizer, we can communicate constraints to the optimizer that would have otherwise potentially required an exponential number of ordinary core constraints.

The structure of the instances $\mathcal{F}^{n,r}$ discussed in Example 2 provides some intuition for abstract cores. In these instances the identity of the variables does not matter, all that matters is how many are set to TRUE and how many are set to FALSE. For example, in any core κ of $\mathcal{F}^{n,r}$ we can exchange any soft clause $C \in \kappa$ for any other soft clause $C' \notin \kappa$ and the result will still be a core of $\mathcal{F}^{n,r}$. In other words, every soft clause is exchangeable with every other soft clause in these instances. While it seems unlikely that complete exchangeability would hold for other instances, it is plausible that many instances might contain subsets of soft clauses that are exchangeable or nearly exchangeable. In particular, in any MaxSat instance the cost of a solution depends only on the number of soft clauses of each weight that it falsifies. The identity of the falsified soft clauses does not matter except to the extent that $\mathcal{F}_H$ might place logical constraints on the set of soft clauses that can be satisfied together.[3]

Abstraction Sets. Suppose we have a set of b-variables all with the same weight and we want to exploit any exchangeability that might exist between their corresponding soft clauses. This can be accomplished by forming an *abstraction set*. An abstraction set, *ab*, is a set of b-variables that have been annotated by adding $|ab|$ new variables, called *ab*'s count variables, used to indicate the number of true b-variables in *ab* (i.e. the number of corresponding falsified soft clauses). The count variables allow us to abstract away from the identity of the particular b-variables that have been made false. We let $ab.c$ denote the sequence of *ab*'s count variables, and let the individual count variables be denoted by $ab.c[1], \ldots, ab.c[|ab|]$. Every count variable has a corresponding definition, with the i'th count variable being defined by the constraint $ab.c[i] \leftrightarrow \sum_{b \in ab} b \geq i$. Note that these definitions can be encoded into CNF and added to the SAT solver using various known encodings for cardinality constraints [6,30,3,27].

Let $\mathcal{AB}$ be a collection of abstraction sets. We require that (1) the sets in $\mathcal{AB}$ are disjoint (so no b-variable is part of two different abstraction sets) and (2) that all of the b-variables in a specific abstraction set $ab \in \mathcal{AB}$ have the same weight (variables in different abstraction sets can have different weights). Let $\mathcal{AB}.c = \bigcup_{ab \in \mathcal{AB}} ab.c$ be the set of all count variables.

[3] This notion of exchangeability is clearly related to symmetries and exploring this connection is a worthwhile direction for future work.

Definition 1. *An* abstract core *is a clause C such that (1) all literals C are either positive b-variables or count variables, $\forall l \in C\,(l \in \mathcal{F}_B \vee l \in \mathcal{AB}.c)$; and (2) C is entailed by $\mathcal{F}_H$ and the conjunction of the count variable definitions, i.e., $\mathcal{F}_H \wedge \left(\bigwedge_{ab.c[k] \in \mathcal{AB}.c} (ab.c[k] \leftrightarrow \sum_{b \in ab} b \geq k\right) \models C$.*

As pointed out in Section 2 every ordinary core is equivalent to a clause containing only positive b-variables that is entailed by $\mathcal{F}_H$. Abstract cores, can be ordinary cores containing only b-variables but they can also contain positive count variables. Like ordinary cores they also must be entailed by $\mathcal{F}_H$ (and the count variable definitions that are required to give meaning to the count variables they contain).

Example 3. Consider an instance $\mathcal{F}^{n,r}$ defined in Example 2. Say we form an single abstraction set, ab, from the full set of blocking variables ${\mathcal{F}_B}^{n,r}$. Then $\mathcal{F}^{n,r}$ will have among its abstract cores the unit clause $(ab.c[r])$ asserting that $\sum_{b \in {\mathcal{F}_B}^{n,r}} b \geq r$. This single abstract core is equivalent to the conjunction of $\binom{n}{n-r+1}$ non-abstract cores. In particular, with n b-variables, asserting that at least r must be true entails that every set of $n - r + 1$ b-variables must contain at least one true b-variable. That is, $(ab.c[r])$ entails $\binom{n}{n-r+1}$ different clauses each of which is equivalent to a non-abstract core. It is not difficult to show that entailment in the other direction also holds giving equivalence.

This example demonstrates the expressive power of abstract cores. More generally, let C be an abstract core containing the count literals $\{ab^1.c[c_1], \ldots, ab^k.c[c_k]\}$. Then, each $ab^i.c[c_i]$ is equivalent to the conjunction of $\binom{|ab^i|}{|ab^i|-c_i+1}$ clauses. Hence, C is equivalent to the conjunction of $\prod_{i=1}^{k} \binom{|ab^i|}{|ab^i|-c_i+1}$ non-abstract cores. In other words, abstract cores achieve the desideratum of providing a compact representation of a large number of cores. We address the second desideratum of being able to reason efficiently with abstract cores in the IHS algorithm in the next section. It can also be noted that core-guided solvers use cardinality constraints and thus are able to generate abstract cores, although they use these cores in a different way than our proposed approach.

5 Abstract Cores in IHS MaxSat solving

Algorithm 3 shows the IHS algorithm extended with abstract cores. Its processing follows the same steps as used earlier in the non-abstract IHS algorithm (Alg. 1). There are however, three changes: (1) the optimizer must now solve a slightly different problem, (2) the abstraction sets are used in `ex-abs-cores` when extracting new constraints for the optimizer and (3) a collection of abstraction sets $\mathcal{AB}$ is maintained and dynamically updated by `update-abs` (line 7). We will assume that `update-abs` is also responsible for updating $\mathcal{F}_H$ so that $\mathcal{F}_H$ always includes all of the count variable definitions, $\bigcup_{ab.c[k] \in \mathcal{AB}.c} \text{CNF}(ab.c[k] \leftrightarrow \sum_{b \in ab} b \geq k)$ as new abstraction sets are added. In this way the other routines given below need only access $\mathcal{F}_H$ assuming that it already includes the count variable definitions.

New Optimization problem: The optimization problem shown in Figure 2a is very similar to the previous minimum cost hitting set optimization (Figure 1a). It continues to minimize the cost of the set of b-variables that have to be set to true in order to satisfy the

```
1  Abstract-IHS (F, wt)
2  LB ← 0; UB ← ∞; τ_best ← ∅
3  C ← ∅; AB ← ∅
4  while true do
5  |  hs ← Min-Abs(F_B, AB, C)
6  |  LB = cost(hs)
7  |  AB ← update-abs(AB, K)
8  |  if (LB = UB) break
9  |  K ← ex-abs-cores
10 |         (hs, AB, UB, τ_best)
11 |  if (LB = UB) break
12 |  C ← C ∪ K
13 return τ_best
```

Algorithm 3: IHS with abstract cores

Min-Abs $(\mathcal{F}_B, \mathcal{AB}, \mathcal{C})$

minimize: $\sum_{b \in \mathcal{F}_B} wt(b) \cdot b$

subject to:

$\sum_{x \in \kappa} x \geq 1$	$\forall \kappa \in \mathcal{C}$
$\sum_{b \in ab} b - k \cdot ab.c[k] \geq 0$	$\forall ab.c[k] \in \mathcal{AB}.c$
$\sum_{b \in ab} b - \lvert ab \rvert \cdot ab.c[k] < k$	$\forall ab.c[k] \in \mathcal{AB}.c$
$b \in \{0, 1\}$	$\forall b \in \mathcal{F}_B$
$ab.c[k] \in \{0, 1\}$	$\forall ab.c[k] \in \mathcal{AB}.c$

return:

$\{b \mid b$ set to 1 in opt. soln$\}$

(a) IP for optimizing with abstract cores

Fig. 2: Adding abstract cores to IHS

constraints. Each abstract core $\kappa \in \mathcal{C}$ is a clause and thus is equivalent to the linear constraint $\sum_{x \in \kappa} x \geq 1$, just like the non-abstract cores. The abstract cores can, however, contain count variables $ab.c[k]$ each of which has a specific definition. These definitions need to be given to the optimizer as linear constraints. For each count variable $ab.c[i]$ the constraints added are (a) $\sum_{b \in ab} b - k \cdot ab.c[k] \geq 0$ and (b) $\sum_{b \in ab} b - |ab| \cdot ab.c[k] < k$. That is, when $ab.c[k]$ is 1 (TRUE) constraint (a) ensures that the sum of *ab*'s b-variables is $\geq k$ and constraint (b) becomes trivial; and when $ab.c[k]$ is 0 (FALSE) constraint (a) becomes trivial and constraint (b) ensures that the sum of *ab*'s b-variables is $< k$. These definitions ensure the intended interaction between abstract cores and count variables. For example, if the optimizer has the abstract core constraint $b_1 + ab.c[5] + b_2 \geq 1$ it must be able to reason that if it chooses to satisfy this constraint by setting $ab.c[5] = 1$ then it must also set 5 of the b-variables in *ab* to 1. The definitions allow this inference.

Extracting Abstract Cores: As before the optimizer's solution is used to create a set of assumptions for the SAT solver. Cores arise from the conflicts the SAT solver finds when using these assumptions. For ordinary cores `ex-cores` (Alg. 2) used a set of negated b-variables as assumptions (ensuring that the corresponding set of soft clauses must be satisfied). If the SAT solver finds a conflict over these assumptions, the conflict will be a clause containing only negated assumptions; i.e, a clause containing only positive b-variables. Such clauses are ordinary cores. Hence, if we wish to extract abstract cores, we must give the SAT solver assumptions that consist of negated b-variables and negated count variables. Any conflicts that arise will then contain positive b-variables and positive count variables and will thus be abstract cores.

In the non-abstract case, the optimizer's solution hs specifies a set of b-variables that can be set to true to obtain an optimal solution to the current set of constraints. That is, hs provides a set of clauses that, if falsified, will most cost effectively block the cores found so far. In the abstract case, the optimizer's solution is also a set of b-

```
1 ex-abs-cores (hs, AB, UB, τ_best)
2     assumps ← {¬b | b ∈ F_B − hs}
3     foreach ab ∈ AB do
4         assumps ← assumps − {¬b | b ∈ ab}
5         if |ab ∩ hs| = |ab| then continue
6         assumps ← assumps ∪ {¬ab.c[|ab ∩ hs| + 1]}
7     K ← ex-cores-sub(assumps, UB, τ_best)
8     optionally: K ← K ∪ ex-cores(hs, UB, τ_best)
9     return K
```

Algorithm 4: Extracting Abstract cores from the optimizer solution

variables with the same properties. All that has changed is the type of constraints the optimizer has optimized over.

Consider an abstraction set in the current set of abstractions $ab \in \mathcal{AB}$. Say that *ab* is the set of b-variables $\{b_1, b_2, b_3, b_4\}$. Further, suppose that the optimizer returns the set $hs = \{b_1, b_4, b_5\}$ as its solution, and that the full set of b-variables is $\mathcal{F}_B = \{b_1, b_2, b_3, b_4, b_5, b_6\}$. In the non-abstract case, the SAT solver will be allowed to make b_1, b_4 and b_5 TRUE, while being forced to make b_2, b_3, and b_6 FALSE. In particular, the SAT solver will be called with the set of assumptions $\neg b_2$, $\neg b_3$ and $\neg b_6$, i.e., the set $\{\neg b \,|\, b \in (\mathcal{F}_B - hs)\}$ (line 7, Alg. 2). Notice, that the SAT solver is being allowed to make specific b-variables in $ab \cap hs$ TRUE (namely b_1 and b_4), while being forced to make specific b-variables in $ab - (ab \cap hs)$ FALSE (namely b_2 and b_3). Given that we believe the b-variables in *ab* to exchangeable, we can achieve abstraction by removing these specific choices. In particular, instead of assuming that b_2 and b_3 are FALSE and forcing the SAT solver to satisfy these specific soft clauses, we can instead assume $\neg ab.c[3]$. This means allowing at most two b-variables in *ab* to be true, forcing the remaining $|ab| - 2\ (= 2)$ b-variables to be FALSE. Hence, the SAT solver must satisfy at least two soft clauses from the set $\{(\neg b_1), (\neg b_2), (\neg b_3), (\neg b_4)\}$ corresponding to *ab*, but it is no longer forced to try to satisfy the specific clauses $(\neg b_2)$ and $(\neg b_3)$. Hence, we can use $\{\neg ab.c[3], \neg b_6\}$ as the SAT solver's assumptions and thus be able to extract an abstract core. Note also that since the weight of every b-variable in *ab* is the same, the SAT solver is still being asked to find a solution of cost equal to $cost(hs)$. Using this insight we can specify the procedure `ex-abs-cores` used to extract abstract cores.

Algorithm 4 shows the procedure `ex-abs-cores`. Once it has set up its assumptions this procedure operates exactly like `ex-cores`, calling the same subroutine `ex-cores-sub` to iteratively extract some number of cores. It first adds the negation of all b-variables not in hs: the optimizer wants to satisfy all of these soft clauses. Then it performs abstraction. It removes the b-variables of each abstraction set *ab* from the assumptions, and adds instead a single count variable from *ab*. The optimizer's solution has made $k = |hs \cap ab|$ of *ab*'s b-variables TRUE. So we permit the SAT solver to make this number of *ab*'s b-variables TRUE, but no more. This is accomplished by giving it the assumption $\neg ab.c[k+1]$. Note that $\neg ab.c[k+1] \leftrightarrow \sum_{b \in ab} b \leq k$ by the definition of the count variables. Finally, if every b-variable of *ab* is in hs we need not add anything to the set of assumptions (line 5): the SAT solver can freely make all of *ab*'s b-variables true.

`ex-abs-cores` also has the option of additionally extracting a set of non-abstract cores by invoking its non-abstract version (line 8). Abstract and non-abstract cores can be freely mixed in `Abstract-IHS`. Due to the indeterminism in the conflicts the SAT solver returns, the non-abstract cores need not be subsumed by the abstract cores. Hence, in practice it is often beneficial to extract both.

The correctness of the IHS algorithm with abstract cores is easily proved.

Theorem 1. *Let $(\mathcal{F}, wt)$ be a MaxSat instance with $\mathcal{F}_H$ being satisfiable and assume that (a) the optimizer correctly returns optimal solutions to its set of constraints, and (b) every conflict C over assumptions returned by the SAT solver is a clause that is entailed by the formula it is solving. Then* `Abstract-IHS` *when called on $(\mathcal{F}, wt)$ must eventually terminate returning an optimal solution.*

Proof. First observe that the extra clauses E used to define the count variables in $\mathcal{AB}.c$ do not change the set of solutions (models of $\mathcal{F}_H$) nor their costs as they are definitions. In particular, any model τ of $\mathcal{F}_H$ can be extended to a model of $\mathcal{F}_H \cup E$ by appropriately setting the value of each count variable, and any model τ^E of $\mathcal{F}_H \cup E$ becomes a model of $\mathcal{F}_H$ once we remove its assignments to the count variables. In both cases the cost of the model is preserved. Therefore, we will prove that `Abstract-IHS` eventually terminates returning an optimal solution to $(\mathcal{F} \cup E, wt)$ (with every clause in E being hard): this optimal solution provides us with an optimal solution to $(\mathcal{F}, wt)$.

From the definitions of the count variables in E and the soundness of the abstract cores computed as assumption conflicts by the SAT solver, we see that every constraint in the optimizer's model is entailed by $\mathcal{F}_H \cup E$. That is, every solution of $\mathcal{F}_H \cup E$ is also a solution of the optimizer's constraints. Therefore, the cost of the optimizer's optimal solutions, LB, is always a lower bound on the cost of an optimal solution of $\mathcal{F}_H \cup E$. Furthermore, τ_{best} is always a solution of $\mathcal{F}_H \cup E$ as it is found by the SAT solver. Therefore, when $UB = cost(\tau_{best}) = LB$, τ_{best} must be an optimal solution. Hence we have that when `Abstract-IHS` returns a solution, that solution must be optimal.

Furthermore, when the optimizer returns a solution hs to its model and hs does not cause termination, then `Abstract-IHS` will compute a new abstract core κ that hs does not satisfy. This follows from the fact κ is falsified by all solutions that make FALSE exactly the same set of un-abstracted b-variables and exactly the same count of b-variables from each abstraction set as hs. Hence, once we add κ to the optimizer we block the solution hs. There are only a finite number of solutions to the optimizer's constraints since the variables all 0/1 variables, and every optimal MaxSat solution of $\mathcal{F}_H \cup E$ always satisfies the optimizer's model. Therefore, as more constraints are added to the optimizer it must eventually return one of these optimal MaxSat solutions causing `Abstract-IHS` to terminate. □

Example 4. Consider running `Abstract-IHS` on the formula used in Example 1: $\mathcal{F}_H = \{(b_1, b_2), (b_2, b_3), (b_3, b_4)\}$, $\mathcal{F}_S = \{(\neg b_1), (\neg b_2), (\neg b_3), (\neg b_4)\}$, and all weights equal to 1. First `Min-Abs` is called on an empty set of constraints, and it returns $hs = \emptyset$. Say that `update-abs` creates a single abstraction set, $\mathcal{AB} = \{ab\}$, with $ab = \{b_2, b_3\}$. Furthermore, say that $\mathcal{AB}$ is unchanged during the rest of the algorithm's run.

Using hs, `ex-abs-cores` will initialize its assumptions to $\{\neg b_1, \neg ab.c[1], \neg b_4\}$ and call the SAT solver. These assumptions are *unsat*. Let the conflict found be the unit

clause $(ab.c[1])$. In `ex-cores-sub` the next SAT call will be with the assumptions $\{\neg b_1, \neg b_4\}$. These assumptions are satisfiable and the solution $\tau = \{\neg b_1, b_2, b_3, \neg b_4\}$ is returned. The upper bound UB will be set to $cost(\tau) = 2$ and τ_{best} will be set to τ.

`ex-abs-cores` now returns and the optimizer is called with the set of abstract cores $\mathcal{C} = \{(ab.c[1])\}$. The optimizer can return two different optimal solutions $\{b_2\}$ or $\{b_3\}$, and say that it returns the first one $hs = \{b_2\}$. This will set the lower bound $LB = cost(hs) = 1$. Then `ex-abs-cores` will be called again and from hs it will initialize its assumptions to $\{\neg b_1, \neg ab.c[2], \neg b_4\}$, which is *unsat* with the unique conflict $(b_1, ab.c[2], b_4)$. Hence, the next SAT call will be with an empty set of assumptions and a solution will be found. Suppose that this solution is the same as before, so that neither UB nor τ_{best} is changed. `ex-abs-cores` will then return and the optimizer called with the accumulated cores $\{(ab.c[1]), (b_1, ab.c[2], b_4)\}$. There are different choices for the optimal solution, but say that it returns $\{b_2, b_3\}$ as its optimal solution. This will reset the lower bound LB to 2, the lower bound will meet the upper bound, and the MaxSat optimal solution $\tau_{best} = \{\neg b_1, b_2, b_3, \neg b_4\}$ will be found.

Abstract cores can decrease the worst-case number of cores the IHS algorithm needs to extract. Consider the instances $\mathcal{F}^{n,r}$ from Examples 2 and 3. As discussed in Example 2 when r is close to $n/2$ these instances have an exponential number of non-abstract cores all of which must be extracted by the IHS algorithm. If on the other hand all b-variables are placed into a single abstraction set *ab* as in Example 3, `Abstract-IHS` will generate the sequence of abstract cores $(ab.c[1]), \ldots, (ab.c[r])$ after which the optimizer will return a solution of cost r that will be a correction sets allowing `Abstract-IHS` to terminate.

More generally, this strategy can be applied to any unweighted MaxSat instance.

Proposition 1. *Let $(\mathcal{F}, wt)$ be an unweighted MaxSat instance, i.e. $w(C) = 1$ for all $C \in \mathcal{F}_S$ and construct an abstraction set ab containing all $\mathcal{F}_B$. Then Algorithm 3 needs to extract at most $|\mathcal{F}_s|$ cores before terminating.*

The solving strategy of Proposition 1 in fact mimics the Linear UNSAT-SAT algorithm [17,9] where the SAT solver solves the sequence of queries "can a solution of cost 1 be found", "can a solution of cost 2 be found", etc. More generally, we note that the framework of abstract cores presented here can be used to mimic the behaviour several of the recently proposed core-guided algorithms [26,25,23].

Computing Abstraction Sets When computing abstraction sets, there is an inherent trade-off between the overhead and potential benefits from abstraction; too large sets can lead to large CNF encodings for the count variable definitions, making SAT solving very inefficient while with too small sets the algorithm reverts back to non-abstract IHS with hardly any gain from abstraction.

Although the notion of exchangeability has intuitive appeal, it seems likely be computationally hard to identify exchangeable b-variables that can be grouped into abstraction sets. In our implementation we used a heuristic approach to finding abstraction sets motivated by the $\mathcal{F}^{n,r}$ instances (Example 1). In those instances, each b-variable appears in many cores with each of the other b-variables. Hence, we decided to build abstraction sets from sets of b-variables that often appear in cores together. This tech-

nique worked in practice, as we will see in Section 6, but a deeper understanding of how best to construct abstraction sets remains as future work.

To find b-variables that appear in many cores with each other, we used the set of cores found to construct a graph G. The graph has b-variables as nodes and weighted edges between two b-variables representing the number of times these two b-variables appeared together in a core. We then applied the Louvain clustering algorithm [10] to G obtain clusters of nodes such that the nodes in a cluster have a higher weight of edges between each other than with nodes in other clusters. Each of these clusters thus represents a heuristically selected set of b-variables that often appear together in cores, these were then taken to define an abstraction set.

We also monitored how effective the cores found were in increasing the lower bound generated by the optimizer. If the cores were failing to drive the optimizer's lower bound higher, we computed new abstraction sets by clustering the graph G, and updated $\mathcal{AB}$ with these new abstraction sets. If clustering had already been performed and the extracted cores were still not effective, the nodes of the b-variables in each abstraction set were merged into one new node and G was reclustered. (The Louvain algorithm can compute hierarchical clusters). Any new clusters so generated will either be new abstraction sets or supersets of existing abstractions sets. New abstraction sets are formed from these new clusters and added to $\mathcal{AB}$. All subsets are removed from $\mathcal{AB}$ so that future abstractions will be generated using the larger abstraction sets.

We also found that that abstraction was not cost effective on instances where the average core size was in the hundreds. The generated abstraction sets were so large that the CNF encoding of their count variables definitions slowed the SAT solver down too much. Finally, only we add the CNF encoding of the count variable definitions $ab.c[k] \leftrightarrow \sum_{b \in ab} b \geq k$ to the SAT solver when $\neg ab.c[k]$ first appears in the set of assumptions. Furthermore, we only add the encoding in the direction $ab.c[k] \leftarrow \sum_{b \in ab} b \geq k$.

6 Experimental Evaluation

We have implemented two versions of our approach on top of the MaxHS solver [14,12] using the version that had been submitted to the MaxSat 2019 evaluation (MSE 2019) [5]. The two new solvers are called **maxhs-abs** and **maxhs-abs-ex**. **maxhs-abs** implements the abstraction method described above, using the Louvain algorithm to dynamically decide on the abstraction sets and extracting both abstract and non-abstract cores in `ex-abs-cores`. We used the well known totalizer encoding [6] to encode the count variable definitions into CNF. In particular, each totalizer takes as input the b-variables of an abstraction set *ab*, and the totalizer outputs become the count variables $ab.c[k]$. We dynamically added the clauses defining the i-th output ($ab.c[k]$) to the SAT solver only when needed.

The **maxhs-abs-ex** solver additionally exploits the totalizer encodings by using the technique of core exhaustion [20]. This technique uses SAT calls to determine a lower bound on the number of totalizer outputs forced to be TRUE. This technique can sometimes force many of the abstraction set count variables. We impose a resource bound of 60s on the process so exhaustion is not complete.

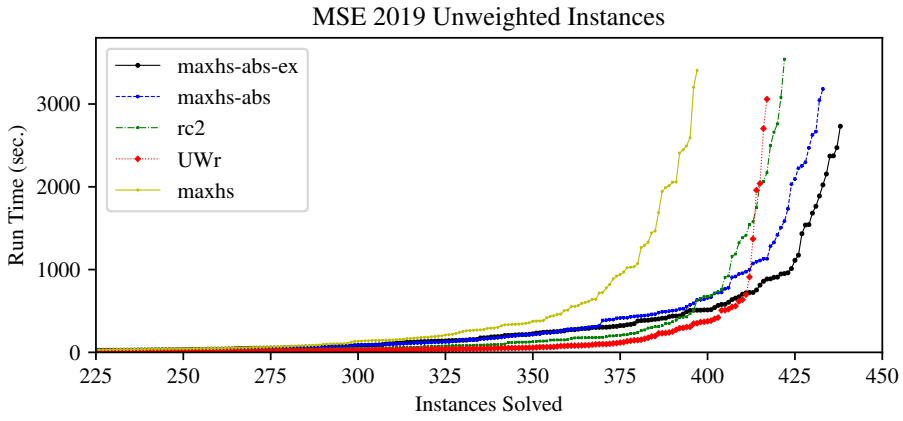

Fig. 3: Cactus plot of solver performance on the 599 MSEval 2019 unweighted instances.

Solver	Unweighed	Weighted	Total
maxhs-abs-ex	**438**	**387**	**825**
maxhs-abs	433	379	812
rc2	422	376	798
UWr	417	368	785
maxhs	397	361	758

Table 1: Number of Solved instances on MSE 2019 instances

We compare the new solvers to the base **maxhs** (MSE 2019 version) as well as to two other solvers: **rc2**, the MSE 2019 version of RC2 [20,4] that was the best performing solver in both the weighted and unweighted track and **UWr**, a new solver in MSE 2019 called UWrMaxSat [21,4]. Both **rc2** and **UWr** implement the OLL algorithm [25,1] and differ mainly in how the cardinality constraints are encoded into CNF. As benchmarks, we used all 599 weighted and 586 unweighted instances from the complete track of the 2019 MaxSat Evaluation, drawn from a variety of different problem families. All experiments were run on a cluster of 2.4 GHz Intel machines using a per-instance time limit of 3600s and memory limit of 5GB.

Figures 3 and 4 show cactus plots comparing the solvers on the unweighted and weighted instances, respectively and Table 1 reports the number of instances solved by each solver within the time and memory limits. Comparing **maxhs** and **maxhs-abs** we observe that abstract core reasoning is very effective, increasing the number of unweighted instances solved from 397 to 433 and weighted instances from 361 to 379 surpassing both **rc2** and **UWr** in both categories. **maxhs-abs-ex** improves even further with 438 unweighted and 387 weighted instances solved surpassing all other solvers.

Table 2 gives a pair-wise solver comparison of the number of instances that could be solved by one solver but not by the other. We observe that even though the solvers can be ranked by number of instances solved, every solver was able to beat every other solver

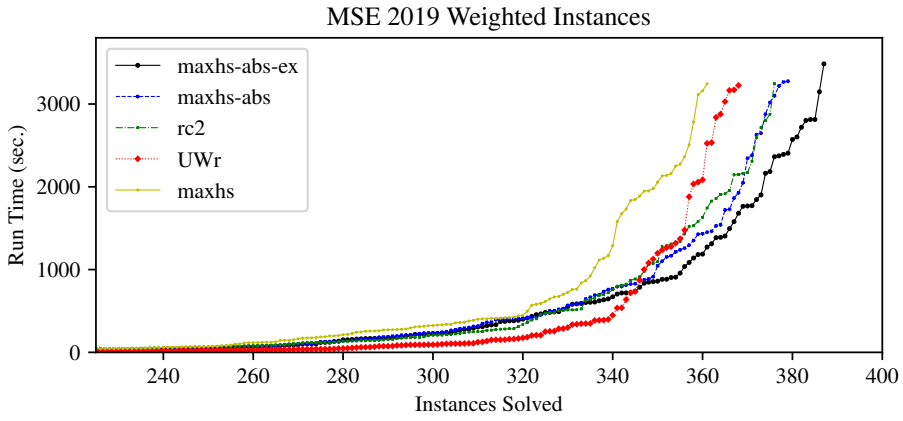

Fig. 4: Cactus plot of solver performance on the 586 MSEval 2019 weighted instances.

Solver	maxhs-abs-ex	maxhs-abs	rc2	UWr	maxhs
maxhs-abs-ex		7/8	26/37	29/46	60/42
maxhs-abs	2/0		27/33	31/43	55/35
rc2	10/26	16/30		12/29	61/50
UWr	8/27	15/32	7/21		61/52
maxhs	19/16	19/17	36/35	41/45	

Table 2: The entry in cell (X, Y) shows the number instances solved by solver X that were not solved by solver Y in the format #Unweighted/#Weighted.

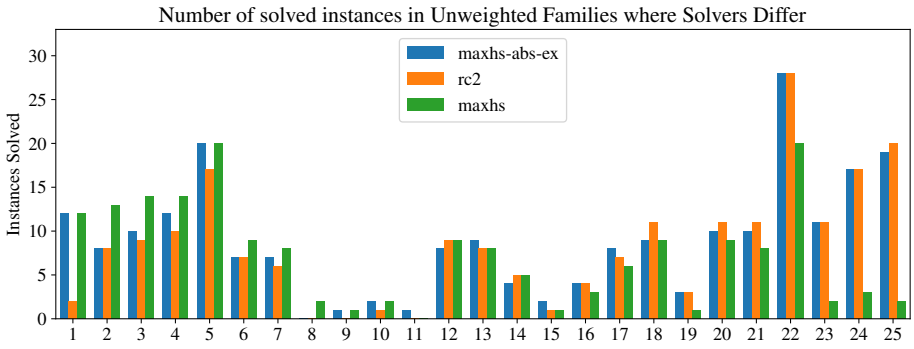

Fig. 5: On the 48 families of Unweighted instances **maxhs-abs-ex**, **rc2**, and **maxhs** solved the same number of instances in 23 families. The number of instances solved in the other 25 families are plotted above. Families on the left are those where **maxhs** solved more instances than **rc2** and those on the right are where **rc2** solved more instances than **maxhs**. The families are plotted with the following indices: 1. kbtree, 2. extension-enforcement, 3. optic, 4. logic-synthesis, 5. close_solutions, 6. min-fill, 7. atcoss, 8. set-covering, 9. maxcut, 10. aes, 11. gen-hyper-tw, 12. frb, 13. bcp, 14. HaplotypeAssembly, 15. scheduling, 16. CircuitTraceCompaction, 17. xai-mindset2, 18. MaxSATQueriesinInterpretableClassifiers, 19. reversi, 20. aes-key-recovery, 21. uaq, 22. MaximumCommonSub-GraphExtraction, 23. protein_ins, 24. drmx-atmostk, 25. fault-diagnosis

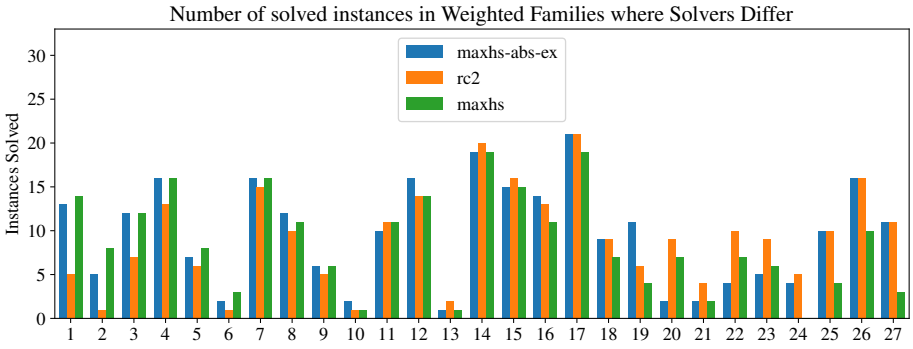

Fig. 6: On the 39 families of Weighted instances **maxhs-abs-ex**, **rc2**, and **maxhs** solved the same number of instances in 12 families. The number of instances solved in the other 27 families are plotted above. Families on the left are those where **maxhs** solved more instances than **rc2** and those on the right are where **rc2** solved more instances than **maxhs**. The families are plotted with the following indices: 1. BTBNSL, 2. maxcut, 3. correlation-clustering, 4. auctions, 5. ParametricRBACMaintenance, 6. ramsey, 7. set-covering, 8. timetabling, 9. relational-inference, 10. hs-timetabling, 11. frb, 12. mpe, 13. railway-transport, 14. metro, 15. max-realizability, 16. MaxSATQueriesinInterpretableClassifiers, 17. haplotyping-pedigrees, 18. drmx-cryptogen, 19. spot5, 20. af-synthesis, 21. min-width, 22. css-refactoring, 23. shiftdesign, 24. lisbon-wedding, 25. tcp, 26. rna-alignment, 27. drmx-atmostk

on some instances (except that **maxhs-abs** did not solve any weighted instances that **maxhs-abs-ex** could not). This speaks to the diversity of the instances, and indicates that truly robust solvers might have to employ a variety of different techniques. As mentioned earlier, in a certain sense abstract cores can be viewed as importing ideas from core-guided solvers into IHS solvers. Its success indicates that other hybridizations might also be useful.

Figures 5 and 6 show a breakdown by family for the three solvers **maxhs-abs-ex**, **rc2** and **maxhs**. The plots show only those families where the solvers exhibited different performance. We observe that **rc2** and **maxhs** achieve quite disparate performance with each one dominating the other on different families. **maxhs-abs-ex**, on the other hand, is often able to achieve the same performance as the better of the two other solvers on these different families. In this sense, it achieves the best of two worlds.

7 Conclusion

We proposed abstract cores for improving the Implicit Hitting Set (IHS) based approach to complete MaxSat solving. More specifically, we address the large worst-case number of cores that IHS needs to extract before terminating. An abstract core is a compact representation of a (potentially large) set of (regular) cores. We show how to incorporate abstract core reasoning into the IHS algorithm, prove correctness of the resulting algorithm and report on an experimental evaluation comparing IHS with abstract cores to the best performing solvers of the latest MaxSat Evaluation. The results indicate that abstract cores indeed improve the empirical performance of the IHS algorithm, resulting in state-of-the-art performance on the instances of the Evaluation. Future work includes exploring more sophisticated methods of building abstraction sets as well as the connection between abstract cores and core guide reasoning further.

References

1. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: Dovier, A., Costa, V.S. (eds.) Technical Communications of ICLP. LIPIcs, vol. 17, pp. 211–221. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012), https://doi.org/10.4230/LIPIcs.ICLP.2012.211
2. Ansótegui, C., Bonet, M.L., Levy, J.: SAT-based MaxSAT algorithms. Artificial Intelligence **196**, 77–105 (2013), http://dx.doi.org/10.1016/j.artint.2013.01.002
3. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks and their applications. In: Kullmann, O. (ed.) Proc SAT. LNCS, vol. 5584, pp. 167–180. Springer (2009), https://doi.org/10.1007/978-3-642-02777-2_18
4. Bacchus, F., Järvisalo, M., Martins, R.: Maxsat evaluation 2018: New developments and detailed results. J. Satisf. Boolean Model. Comput. **11**(1), 99–131 (2019), https://doi.org/10.3233/SAT190119
5. Bacchus, F., Järvisalo, M., Martins, R. (eds.): MaxSAT Evaluation 2019: Solver and Benchmark Descriptions. Department of Computer Science Report Series B, Department of Computer Science, University of Helsinki, Finland (2019)
6. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In: Rossi, F. (ed.) Proc CP. LNCS, vol. 2833, pp. 108–122. Springer (2003), https://doi.org/10.1007/978-3-540-45193-8_8
7. Berg, J., Hyttinen, A., Järvisalo, M.: Applications of maxsat in data analysis. In: Berre, D.L., Järvisalo, M. (eds.) Proc Pragmatics of SAT. EPiC Series in Computing, vol. 59, pp. 50–64. EasyChair (2018), http://www.easychair.org/publications/paper/6HpF
8. Berg, J., Järvisalo, M.: Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. Artif. Intell. **244**, 110–142 (2017), https://doi.org/10.1016/j.artint.2015.07.001
9. Berre, D.L., Parrain, A.: The Sat4j library, release 2.2. J. Satisf. Boolean Model. Comput. **7**(2-3), 59–6 (2010)
10. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), P10008 (Oct 2008), https://doi.org/10.1088%2F1742-5468%2F2008%2F10%2Fp10008
11. Chen, Y., Safarpour, S., Marques-Silva, J., Veneris, A.G.: Automated design debugging with maximum satisfiability. IEEE Trans. on CAD of Integrated Circuits and Systems **29**(11), 1804–1817 (2010), https://doi.org/10.1109/TCAD.2010.2061270
12. Davies, J.: Solving MAXSAT by Decoupling Optimization and Satisfaction. Ph.D. thesis, University of Toronto (2013)
13. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Lee, J.H. (ed.) Proc CP. LNCS, vol. 6876, pp. 225–239. Springer (2011), https://doi.org/10.1007/978-3-642-23786-7_19
14. Davies, J., Bacchus, F.: Exploiting the power of mip solvers in maxsat. In: Järvisalo, M., Gelder, A.V. (eds.) Proc SAT. LNCS, vol. 7962, pp. 166–181. Springer (2013), https://doi.org/10.1007/978-3-642-39071-5_13
15. Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: Schulte, C. (ed.) Proc CP. LNCS, vol. 8124, pp. 247–262. Springer (2013), https://doi.org/10.1007/978-3-642-40627-0_21
16. Demirovic, E., Musliu, N., Winter, F.: Modeling and solving staff scheduling with partial weighted maxsat. Annals OR **275**(1), 79–99 (2019), https://doi.org/10.1007/s10479-017-2693-y
17. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. J. Satisf. Boolean Model. Comput. **2**(1-4), 1–26 (2006), https://satassociation.org/jsat/index.php/jsat/article/view/18

18. Ghosh, B., Meel, K.S.: IMLI: an incremental framework for maxsat-based learning of interpretable classification rules. In: Conitzer, V., Hadfield, G.K., Vallor, S. (eds.) Proc AIES. pp. 203–210. ACM (2019), https://doi.org/10.1145/3306618.3314283
19. Hosokawa, T., Yamazaki, H., Misawa, K., Yoshimura, M., Hirama, Y., Arai, M.: A low capture power oriented x-filling method using partial maxsat iteratively. In: Proc IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT. pp. 1–6. IEEE (2019), https://doi.org/10.1109/DFT.2019.8875434
20. Ignatiev, A., Morgado, A., Marques-Silva, J.: RC2: an efficient maxsat solver. J. Satisf. Boolean Model. Comput. **11**(1), 53–64 (2019), https://doi.org/10.3233/SAT190116
21. Karpinski, M., Piotrów, M.: Encoding cardinality constraints using multiway merge selection networks. Constraints **24**(3-4), 234–251 (2019), https://doi.org/10.1007/s10601-019-09302-0
22. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: Qmaxsat: A partial max-sat solver. J. Satisf. Boolean Model. Comput. **8**(1/2), 95–100 (2012), https://satassociation.org/jsat/index.php/jsat/article/view/98
23. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. CoRR **abs/0712.1097** (2007)
24. Martins, R., Manquinho, V.M., Lynce, I.: Open-wbo: A modular maxsat solver,. In: Sinz, C., Egly, U. (eds.) Proc SAT. LNCS, vol. 8561, pp. 438–445. Springer (2014), https://doi.org/10.1007/978-3-319-09284-3_33
25. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided maxsat with soft cardinality constraints. In: O'Sullivan, B. (ed.) Proc CP. LNCS, vol. 8656, pp. 564–573. Springer (2014), https://doi.org/10.1007/978-3-319-10428-7_41
26. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided maxsat resolution. In: Brodley, C.E., Stone, P. (eds.) Proc AAAI. pp. 2717–2723. AAAI Press (2014), http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8513
27. Ogawa, T., Liu, Y., Hasegawa, R., Koshimura, M., Fujita, H.: Modulo based CNF encoding of cardinality constraints and its application to maxsat solvers. In: Proc ICTAI. pp. 9–17. IEEE Computer Society (2013), https://doi.org/10.1109/ICTAI.2013.13
28. Saikko, P.: Re-implementing and Extending a Hybrid SAT-IP Approach to Maximum Satisfiability. Master's thesis, University of Helsinki (2015), http://hdl.handle.net/10138/159186
29. Saikko, P., Berg, J., Järvisalo, M.: LMHS: A SAT-IP hybrid maxsat solver. In: Creignou, N., Berre, D.L. (eds.) Proc SAT. LNCS, vol. 9710, pp. 539–546. Springer (2016), https://doi.org/10.1007/978-3-319-40970-2_34
30. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: van Beek, P. (ed.) Proc CP. LNCS, vol. 3709, pp. 827–831. Springer (2005), https://doi.org/10.1007/11564751_73
31. Zhang, L., Bacchus, F.: MAXSAT heuristics for cost optimal planning. In: Hoffmann, J., Selman, B. (eds.) Proc AAAI. AAAI Press (2012), http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5190