Abstract Cores in Implicit Hitting Set MaxSat Solving
(Extended Abstract)

Jeremias Berg, Fahiem Bacchus, Alex Poole
1University of Helsinki, HIIT, Department of Computer Science, Finland
2University 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 for solving MaxSat instances from real-world domains are the so-called implicit hitting set (IHS) solvers. IHS solvers decouple MaxSat solving into separate core-extraction (i.e., reasoning) and optimization steps which are tackled by a Boolean satisfiability (SAT) and an integer linear programming (IP) solver, respectively. While the approach shows state-of-the-art performance on many industrial instances, it is known that there exists instances on which IHS solvers need to extract an exponential number of cores before terminating. Motivated by the simplest of these problematic instances, we propose abstract cores, 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 2019 MaxSat Evaluation.

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 [Bacchus et al., 2021]. Effective MaxSat solvers enable solving instances of NP-hard optimisation problems by encoding them into propositional formulas in conjunctive normal form (CNF) and then computing an assignment to the variables in the formula that maximizes the number (or weight) of satisfied constraints. Algorithmically MaxSat is typically treated as the equivalent problem of minimizing the weight of unsatisfied constraints.

As witnessed by the annual MaxSAT evaluations [Bacchus et al., 2019a], the currently most successful complete MaxSAT solvers rely heavily on the exceptionally effective Boolean satisfiability (SAT) solvers. SAT solvers are usually used in order to iteratively extract either: (i) MaxSAT solutions of improving quality [Koshimura et al., 2012], or (ii) unsatisfiable cores, i.e., small sets containing constraints that can not all be satisfied by a single assignment. Pure SAT-based solvers that extract cores [Ansótegui et al., 2013] then relax the formula in order to allow, in a controlled way, an increasing number of constraints to be falsified in subsequent iterations. The so-called implicit hitting set (IHS) solvers [Bacchus et al., 2019a; Davies and Bacchus, 2013a; Saikko et al., 2016] instead give the cores to an Integer Linear Programming (ILP) optimizer that computes minimum-cost subsets of constraints to be considered for falsification in subsequent iterations. IHS solvers are currently one of the most successful approaches to solving MaxSat instances encountered in practical applications [Bacchus et al., 2019a].

By decoupling MaxSat solving into separate core extraction and optimisation steps, IHS solvers are able to exploit the disparate strengths of SAT and IP solvers and avoid increasing the complexity of the underlying SAT instance by deferring all numerical reasoning to the optimizer [Davies and Bacchus, 2011]. One drawback of the approach, however, is that on some formulas, an exponential number of cores need to be extracted by the SAT solver. In this work we analyze the simplest formulas exhibiting this exponential worse case and propose abstract cores as a compact representation for a potentially exponential number of ordinary cores. We show how the IHS algorithm can be extended to support reasoning over abstract cores and that by doing so we can in principle achieve an exponential reduction in the number of constraints the SAT solver has to extract and give to the optimizer.

This abstract is based on [Berg et al., 2020]. Here we overview the IHS algorithm for MaxSat, informally introduce the concept of abstract cores and discuss how the IHS algorithm can be extended with abstract cores, both in theory and practice. We also demonstrate empirically that adding abstract core reasoning to a state-of-the-art IHS solver improves its performance enough to surpass the best performing solvers of the 2019 MaxSat evaluation.

2 Maximum Satisfiability
For a Boolean variable \( x \) there are two literals, the positive \( x \) and the negative \( \neg x \). A CNF formula \( \mathcal{F} \) is a conjunction of clauses, each of which is a disjunction of literals. We mostly
treat $\mathcal{F}$ and $C$ as sets of clauses and literals, respectively. Hence we write clauses $C = (x \lor y \lor \neg z)$ in set notation by $\{x, y, \neg z\}$ and $C \in \mathcal{F}$ to indicate that the clause $C$ is in the formula $\mathcal{F}$. An assignment $\tau$ that maps variables to 1 (TRUE) or 0 (FALSE) satisfies $C (\tau(C) = 1)$ if it assigns a positive literal in $C$ to TRUE or a negative literal in $C$ to FALSE. A formula $\mathcal{F}$ is satisfied by $\tau (\tau(\mathcal{F}) = 1)$ if all of its clauses are satisfied. Such $\tau$ is a model of $\mathcal{F}$. A formula is satisfiable if it has a model, otherwise it is unsatisfiable.

An instance $I = (\mathcal{F}_H, \mathcal{F}_S, wt)$ of (weighted partial) MaxSat consists of two CNF formulas, the hard clauses $\mathcal{F}_H$, the soft clauses $\mathcal{F}_S$, and a weight function $wt$ that maps every soft clause $C \in \mathcal{F}_S$ to an integer weight $wt(C) > 0$. The instance is unsatisfiable if $wt(C_1) = wt(C_2)$ holds for any two $C_1, C_2 \in \mathcal{F}_S$. An assignment $\tau$ is a solution to $I$ if it satisfies the hard clauses. The cost $cost(I, \tau) = \sum_{C \in \mathcal{F}_S} (1 - \tau(C)) wt(C)$ of a solution $\tau$ is the sum of the weights of soft clauses it falsifies. When the instance is clear from context, we shorten notation to $cost(\tau)$. A solution $\tau$ to $I$ is optimal if $cost(\tau) \leq cost(\tau^*)$ holds for all solutions $\tau^*$ to $I$. The cost of an instance, $cost(I)$, is the cost of its optimal solutions. The objective of MaxSAT is to compute a (any) optimal solution to $I$. To simplify notation, we will assume that every soft clause $C \in \mathcal{F}_S$ is a unit negative literal, i.e. $C = (\neg b)$ for a variable $b$. The assumption can be made w.l.o.g.; for any soft clause $D \in \mathcal{F}_S$ we can take a new variable $b$ and transform $D$ into the hard clause $(D \lor \neg b)$ and soft clause $(\neg b)$ with $wt((\neg b)) = wt(C)$. We say that a variable $b$ for which $(\neg b) \in \mathcal{F}_S$ is a blocking variable (b-variable) of $I$ and denote the set of all b-variables of $I$ by $\mathcal{B}_I$.

The implicit hitting set based algorithm for computing an optimal solution to a MaxSat instance $I = (\mathcal{F}_H, \mathcal{F}_S, wt)$ makes extensive use of (unsatisfiable) cores and hitting sets. A set $k \subset \mathcal{F}_S$ is a core if $\mathcal{F}_H \wedge k$ is unsatisfiable. For our work it is convenient to view cores as clauses over (or sets of) positive b-variables that are entailed by $\mathcal{F}_H$. Since every clause $C \in k$ is a unit clause containing the negation of a b-variable $b \in \mathcal{B}_I$ and assigning $b = 1$ corresponds to falsifying $C$, it follows from $\mathcal{F}_H \wedge k$ being unsatisfiable that every model of $\mathcal{F}_H$ (i.e. solution of $I$) also satisfies the clause $\{b \mid (\neg b) \in k\}$. Given a collection $\mathcal{C}$ of cores, represented as sets of b-variables, a hitting set $hs \subset \mathcal{B}_I$ over $\mathcal{C}$ is a set of b-variables s.t. $hs \cap \mathcal{C} \neq \emptyset$ for every $\mathcal{C} \in \mathcal{C}$. The cost of a hitting set, $cost(hs) = \sum_{b \in hs} wt((\neg b))$, is the sum of weights of the soft clauses corresponding to the blocking variables in $hs$. $hs$ is minimum-cost if $cost(hs) \leq cost(hs^*)$ holds for every hitting set $hs^*$ over $\mathcal{C}$. As every core in $\mathcal{C}$ is satisfied by any solution to $I$, it is fairly easy to show that $cost(hs) \leq cost(I)$, i.e. that minimum-cost hitting sets over $\mathcal{C}$ provide lower bounds on the optimal cost of $I$. In fact, if $\mathcal{C}$ contains all cores of $I$, then a minimum-cost hitting set $hs$ over $\mathcal{C}$ has $cost(I) = cost(hs) = cost(\tau)$ for a solution $\tau$ that sets $\tau(b) = 1$ for every $b \in hs$ (falsifying $(\neg b)$) and $\tau(b) = 0$ (satisfying $(\neg b) \in \mathcal{F}_S$) for every other $b \in \mathcal{B}_I \setminus hs$.

Example 1 Consider the 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 containing $n - r + 1$ soft clauses is a core.

3 Implicit Hitting Sets for MaxSAT

Algorithm 1 details IHS, the implicit hitting set approach to computing an optimal solution to a MaxSat instance $I$. During search, the algorithm maintains a lower bound $LB$ (initialized to 0), and an upper bound $UB$ (initialized to $\infty$) on $cost(I)$. The lower bound is iteratively refined by computing minimum-cost hitting sets over a set $\mathcal{C}$ of cores of $I$ (initialized to $\emptyset$). The upper bound and set of cores are refined by using a SAT solver to extract cores and solutions of $I$. IHS also maintains a witness for the upper bound in the form of an assignment $\tau_{best}$ for which $cost(\tau_{best}) = UB$. The algorithm terminates when $LB = UB$ and returns $\tau_{best}$.

More specifically, after initialisation (on Lines 2-3) IHS starts its main search loop (Lines 4-11). During each iteration of the loop, a minimum-cost hitting set $hs$ over the current set of cores $\mathcal{C}$ is computed by an ILP optimizer $\text{Min-Hs}$. The hitting set is used to refine the lower bound $LB$ on $cost(I)$ on Line 6. Afterwards, the procedure $\text{ex-cores}$ uses a SAT solver to extract a disjoint set $K$ of previously unseen cores of $I$, i.e. cores for which $\{b_i \in \mathcal{B}_I \setminus hs \mid b_i \cap k_j = \emptyset\}$ for any $k_i, k_j \in K$. $\text{ex-cores}$ terminates when no more such cores can be found and returns the set $K$ and a solution $\tau$ to $I$ that satisfies (at least) all soft clauses that are not in $hs$ nor in any of the computed cores. The current upper bound $UB$ is then compared with $cost(\tau)$ and updated if needed (Line 9).

A detailed description of IHS and its sub-procedures, including a detailed proof of correctness, can be found in [Bacchus et al., 2017]. Our implementation of the base algorithm is extended in a variety of previously proposed ways to allow extracting large numbers (hundreds) of cores from each optimizer solution [Davies and Bacchus, 2013b; Davies, 2013; Saikko, 2015].

4 Abstract Cores

While IHS has been shown to be an effective approach for solving MaxSat instances from real-world applications, the following example demonstrates that there are some drawbacks with it.
Example 2 Consider $F^{n,r}$ from Example 1 and let $C$ be the collection of subsets of $F_B^{n,r}$ that contain exactly $n - r + 1$ variables. All such sets are cores of $F^{n,r}$. From the results of [Davies, 2013] we have that if the optimizer $\text{Min-Hs}$ is given all cores in $C$ it would compute a solution $hs$ with cost$(hs) = r$; furthermore, if even one core of $C$ is missing, the optimizer solutions $hs$ would have cost$(hs) < r$. This means that IHS will have to extract $\binom{n-r+1}{r}$ cores for the optimizer before it can reach the cost of $F^{n,r}$ and terminate. When $r$ is close to $n/2$ the number of cores required for termination is thus exponential in $n$.

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. The results of the 2019 MaxSat Evaluation [Bacchus et al., 2019a; Bacchus et al., 2019b] witness this drawback in practice. The drmx-atmostk instances with $n=11$ were able to solve all 11 instances in under 10s.

A natural question to ask is whether or not there exists a more compact representation of a 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 in the ex-cores procedure, we can communicate constraints to the optimizer that could otherwise have required an exponential number of ordinary cores.

The structure of the $F^{n,r}$ instance from Example 1 provides 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 $\kappa$ of $F^{n,r}$ we can exchange any soft clause $C$ for any other soft clause $C' \notin \kappa$ and the result will still be a core of $F^{n,r}$. In other words, every soft clause is exchangeable with every other soft clause in these instances. The concept of an abstraction set allows abstracting away the specific identities of soft clauses and reason only over the number of them to set to true.

Consider an instance $I = \langle F_H, F_S, wt \rangle$. An abstraction set of $I$ is a subset $ab \subseteq F_B$ of $b$-variables that has been annotated by adding $|ab|$ new variables $ab.c[1], \ldots, ab.c[|ab|]$, 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). Every count variable $ab.c[k]$ has a corresponding definition $ab.c[k] \iff \sum_{b \in ab} b \geq k$. Note that these definitions can be encoded into $\text{CNF}$ using various known encodings for cardinality constraints [Sinz, 2005; Asín et al., 2009] or expressed as the linear constraints $\sum_{b \in ab} b - k \cdot ab.c[k] \geq 0$ and $\sum_{b \in ab} b - (ab) \cdot ab.c[k] < k$. The first constraint ensures that any assignment setting $ab.c[k] = 1$ also sets at least $k$ of the variables in $ab$ to 1. The second constraint ensures that an assignment setting at least $k$ of the variables in $ab$ to 1, also sets $ab.c[k]$ to 1.

An abstract core of $I$ is a clause containing either positive $b$-variables or positive count variables that is entailed by the hard clauses $F_H$ together with the definitions required to give meaning to the count variables.

Example 3 Consider the instance $F^{n,r}$ defined in Example 1. Say we form an single abstraction set, $ab$, from the full set of blocking variables $F_B^{n,r}$. Then $F^{n,r}$ will have among its abstract cores the unit clause $(ab.c[\kappa])$ asserting that $\sum_{b \in F_B^{n,r}} b \geq \kappa$. This single abstract core is equivalent to the conjunction of $\binom{n-r+1}{r}$ 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 $\binom{n-r+1}{r}$ 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. Let $C$ be an abstract core containing the count literals $(ab.c[i], \ldots, ab.c[|ab|])$. Then, each $ab.c[i]$ is equivalent to the conjunction of $\binom{|ab|}{i-1}$ clauses. Hence, $C$ is equivalent to the conjunction of $\prod_{i=1}^{|ab|} \binom{|ab|}{i-1}$ non-abstract cores. In other words, abstract cores achieve the desideratum of providing a compact representation of a large number of cores. Next we address the second aim of extending the IHS algorithm with abstract core reasoning.

5 Abstract Cores in IHS MaxSAT Solving

As the second main contribution of our work, we demonstrate how to efficiently reason with abstract cores in the IHS algorithm. When solving an instance $I = \langle F_H, F_S, wt \rangle$, Abstract-IHS, our extension of IHS with abstract cores differs, from Algorithm 1 in three ways: (1) the optimisation problem that is solved by the optimizer (Min-Hs) is slightly different, (2) the ex-cores subroutine uses abstraction sets when extracting new constraints for the optimizer and (3) a collection of abstraction sets $AB$ is maintained and dynamically updated during search. We briefly overview these differences, more details can be found in [Berg et al., 2020].

New optimisation problem. In Abstract-IHS, the optimizer is given a set $C$ containing both abstract and regular cores as well as the definitions of count variables. The procedure computes a hitting set $hs$ of $b$-variables s.t. for every (abstract) core $\kappa \subseteq C$ the set $hs$ either: (i) contains a $b$-variable $b \in \kappa$ or (ii) contains at least $k$ $b$-variables from $ab$ for a count variable $ab.c[k] \in \kappa$. In practice, this is achieved by adding the definitions of the count variables as linear constraints into the integer program that is solved by Min-Hs.

Extraction of abstract cores. In IHS, new cores are extracted using the assumption interface of a SAT solver [Eén and Sörensson, 2003]. More specifically, the solver is invoked on the clauses in $F_H$ with a set $A$ of assumptions containing $b$-variables that are not in the current hitting $hs$. If the result is unsatisfiable, the solver returns a subset $\kappa \subseteq \{b \mid \neg b \in A\}$ of negated assumptions (i.e. positive $b$-variables) that explain unsatisfiability. This set corresponds to a core of $I$. Thus, in order to be able to extract abstract cores, the SAT
solving the SAT solver needs to be invoked on $F_H$ and the definitions of count variables with a set of assumptions containing negations of count variables and b-variables. Adding the definitions of variables to the SAT solver is straightforward using any of the existing CNF encodings [Bailleux and Boufkhad, 2003; Sinz, 2005; Asín et al., 2009; Ogawa et al., 2013]. As for the assumptions, given a hitting set $h$ we iterate over the set $AH$. For each $ab \in AH$ we remove the variables in $ab$ from $h$ and add the negation of the corresponding count variable $ab.c[[ab \land h]] + 1$ instead. After doing this for every $ab \in AH$, we obtain a set of assumptions that can be used in order to extract abstract cores that are not satisfied by $h$.

Refining the Abstraction Sets

When it comes to computing abstraction sets, there is an inherent trade-off between the overhead and potential benefits from abstraction. In theory, we can show that for any unweighted instance $L$, there exists a (large) abstraction set $ab$ that allows Abstract-IHS to terminate after extracting a polynomial number of cores (recall Example 2 demonstrating that IHS might need an exponential number of cores). However, in practice, too large sets can lead to large CNF encodings of the count variable definitions, making core extraction very inefficient. With too small sets, the algorithm instead reverts back to non-abstract IHS.

In our implementation, we maintain a collection $AH$ of disjoint abstraction sets s.t. any two b-variables in the same abstraction set have the same weight. The abstraction sets are computed by clustering a graph $G$ that has the b-variables as nodes and a weighted edge between two nodes corresponding to b-variables $b_1$ and $b_2$ for which $wt(b_1) = wt(b_2)$. The weight of the edge is the number of times that a core containing both $b_1$ and $b_2$ has been extracted. Intuitively, this procedure aims to create abstraction sets containing b-variables that often appear in cores together, and are thus in some sense related to each other. For clustering $G$ we used the Louvain algorithm [Blondel et al., 2008]. During solving, we monitor the quality of the current abstraction sets. If the found cores are unable to drive up the lower bound computed by the optimizer, we merge all nodes that currently belong to the same cluster (abstraction set), and recluster $G$.

6 Experimental Evaluation

We have implemented two versions of our approach on top of the version of the MaxHS solver [Davies and Bacchus, 2013a; Davies, 2013] submitted to the MaxSat 2019 evaluation (MSE 2019). The two new solvers are called maxhs-abs and maxhs-abs-ex. maxhs-abs implements the abstraction method described in Section 5, using the well known totalizer encoding [Bailleux and Boufkhad, 2003] to encode the count variable definitions into CNF. The maxhs-abs-ex solver additionally implements technique of core exhaustion [Ignatiev et al., 2019] that uses SAT calls to potentially fix count variables from each abstraction set to TRUE.

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 [Ignatiev et al., 2019] that was the best performing solver in both the weighted and unweighted track and UWr, a new solver in MSE 2019 called UWrMaxSat [Karpinski and Piotrów, 2019]. Both rc2 and UWr implement the OLL algorithm [Morgado et al., 2014; Andres et al., 2012] 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 1 shows a cactus plots comparing the solvers on the unweighted and weighted instances, respectively. 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. The potential of abstract cores is further demonstrated by the results of the 2020 MSE where maxhs-abs-ex was the best and second-best performing solver in the unweighted and weighted category, respectively.

7 Conclusions

We proposed abstract cores for improving the IHS based approach to complete MaxSat solving, addressing the large worst-case number of cores that IHS solvers need to extract before terminating. We show how to incorporate abstract core reasoning into the IHS 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.
References


