Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)
Sister Conferences Best Papers Track

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

Jeremias Berg[1] , **Fahiem Bacchus**[2] , **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 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 $C = (x \vee y \vee \neg z)$ 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 τ 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 τ ($\tau(\mathcal{F}) = 1$) if all of its clauses are satisfied. Such τ is a model of $\mathcal{F}$. A formula is satisfiable if it has a model, otherwise it is unsatisfiable.

An instance $\mathcal{I} = (\mathcal{F}_H, \mathcal{F}_S, wt)$ 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 unweighted if $wt(C_1) = wt(C_2)$ holds for any two $C_1, C_2 \in \mathcal{F}_S$. An assignment τ is a solution to $\mathcal{I}$ if it satisfies the hard clauses. The cost $cost(\mathcal{I}, \tau) = \sum_{C \in \mathcal{F}_S}(1 - \tau(C))wt(C)$ of a solution τ 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 τ to $\mathcal{I}$ is optimal if $cost(\tau) \leq cost(\tau^*)$ holds for all solutions τ^* to $\mathcal{I}$. The cost of an instance, $cost(\mathcal{I})$, is the cost of its optimal solutions. The objective of MaxSAT is to compute a (any) optimal solution to $\mathcal{I}$. 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 $(D \vee b)$ 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 $\mathcal{I}$ and denote the set of all b-variables of $\mathcal{I}$ by $\mathcal{F}_B$.

The implicit hitting set based algorithm for computing an optimal solution to a MaxSAT instance $\mathcal{I} = (\mathcal{F}_H, \mathcal{F}_S, wt)$ makes extensive use of (unsatisfiable) cores and hitting sets. A set $\kappa \subset \mathcal{F}_S$ is a core if $\mathcal{F}_H \wedge \kappa$ 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 $C \in \kappa$ is a unit clause containing the negation of a b-variable $b \in \mathcal{F}_B$ and assigning $b = 1$ corresponds to falsifying C, it follows from $\mathcal{F}_H \wedge \kappa$ being unsatisfiable that every model of $\mathcal{F}_H$ (i.e. solution of $\mathcal{I}$) also satisfies the clause $\{b \mid (\neg b) \in \kappa\}$. Given a collection $\mathcal{C}$ of cores, represented as sets of b-variables, a hitting set $hs \subset \mathcal{F}_B$ over $\mathcal{C}$ is a set of b-variables s.t. $hs \cap \kappa \neq \emptyset$ for every $\kappa \in \mathcal{C}$. 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 $\mathcal{I}$, it is fairly easy to show that $cost(hs) \leq cost(\mathcal{I})$, i.e. that minimum-cost hitting sets over $\mathcal{C}$ provide lower bounds on the optimal cost of $\mathcal{I}$. In fact, if $\mathcal{C}$ contains *all* cores of $\mathcal{I}$, then a minimum-cost hitting set hs over $\mathcal{C}$ has $cost(\mathcal{I}) = cost(hs) = cost(\tau)$ for a solution τ 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 $b \in \mathcal{F}_B \setminus hs$.

Example 1 *Consider the instance* ${\mathcal{F}_H}^{n,r} = \text{CNF}(\sum_{i=1}^{n} b_i \geq r)$ *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.*

Algorithm 1: The IHS approach to MaxSat solving

```
1  IHS I = (F_H, F_S, , wt)
       Input: A MaxSat instance I = (F_H, F_S, 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, τ) ← ex-cores (F_H, F_B, hs);
9          if (UB > cost(τ)) τ_best ← τ; UB ← cost(τ);
10         if (LB = UB) break;
11         C ← C ∪ K
12     return τ_best
```

3 Implicit Hitting Sets for MaxSAT

Algorithm 1 details IHS, the implicit hitting set approach to computing an optimal solution to a MaxSAT instance $\mathcal{I}$. During search, the algorithm maintains a lower bound LB (initialized to 0), and an upper bound UB (initialized to ∞) on $cost(\mathcal{I})$. The lower bound is iteratively refined by computing minimum-cost hitting sets over a set $\mathcal{C}$ of cores of $\mathcal{I}$ (initialized to $\emptyset$). The upper bound and set of cores are refined by using a SAT solver to extract cores and solutions of $\mathcal{I}$. IHS also maintains a witness for the upper bound in the form of an assignment τ_{best} for which $cost(\tau_{best}) = UB$. The algorithm terminates when $UB = LB$ and returns τ_{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 Min-Hs. The hitting set is used to refine the lower bound LB on $cost(\mathcal{I})$ on Line 6. Afterwards, the procedure ex-cores uses a SAT solver to extract a disjoint set K of previously unseen cores of $\mathcal{I}$, i.e. cores for which $\kappa_i \subset \mathcal{F}_B \setminus hs$ and $\kappa_i \cap \kappa_j = \emptyset$ for any $\kappa_i, \kappa_j \in K$. ex-cores terminates when no more such cores can be found and returns the set K and a solution τ to $\mathcal{I}$ 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 $\mathcal{F}^{n,r}$ from Example 1 and let $\mathcal{C}$ be the collection of subsets of ${\mathcal{F}_B}^{n,r}$ that contain exactly $n-r+1$ variables. All such sets are cores of $\mathcal{F}^{n,r}$. From the results of [Davies, 2013] we have that if the optimizer* Min-Hs *is given all cores in $\mathcal{C}$ it would compute a solution hs with $cost(hs) = r$; furthermore, if even one core of $\mathcal{C}$ is missing, the optimizer solutions hs would have $cost(hs) < r$. This means that* IHS *will have to extract $\binom{n}{n-r+1}$ cores for the optimizer before it can reach the cost of $\mathcal{F}^{n,r}$ 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 set of instances in the evaluation contain 11 instances with the same underlying structure as Example 1. Out of these, the IHS solver MaxHS [Davies and Bacchus, 2013a; Davies and Bacchus, 2011], 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.

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 $\mathcal{F}^{n,r}$ 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 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. The concept of an *abstraction set* allows abstracting away form the specific identities of soft clauses and reason only over the number of them set to TRUE.

Consider an instance $\mathcal{I} = (\mathcal{F}_H, \mathcal{F}_S, wt)$. An **abstraction set** of $\mathcal{I}$ is a subset $ab \subset \mathcal{F}_B$ 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 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 $\mathcal{I}$ is a clause containing either positive b-variables or positive count variables that is entailed by the hard clauses $\mathcal{F}_H$ together with the definitions required to give meaning to the count variables.

Example 3 *Consider the instance $\mathcal{F}^{n,r}$ defined in Example 1. 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. 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. 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 $\mathcal{I} = (\mathcal{F}_H, \mathcal{F}_S, wt)$, 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 $\mathcal{AB}$ 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 $\mathcal{C}$ 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 $\kappa \in \mathcal{C}$ 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 $\mathcal{F}_H$ with a set $\mathcal{A}$ of assumptions containing b-variables that are not in the current hitting hs. If the result is unsatisfiable, the solver returns a subset $\kappa \subset \{b \mid \neg b \in \mathcal{A}\}$ of negated assumptions (i.e. positive b-variables) that explain unsatisfiability. This set corresponds to a core of $\mathcal{I}$. Thus, in order to be able to extract abstract cores, the SAT

solver needs to be invoked on $\mathcal{F}_H$ 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 hs we iterate over the set $\mathcal{AB}$. For each $ab \in \mathcal{AB}$ we remove the variables in ab from hs and add the negation of the corresponding count variable $ab.c[|ab \cap hs| + 1]$ instead. After doing this for every $ab \in \mathcal{AB}$, we obtain a set of assumptions that can be used in order to extract abstract cores that are not satisfied by hs.

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 $\mathcal{I}$, 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 $\mathcal{AB}$ 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

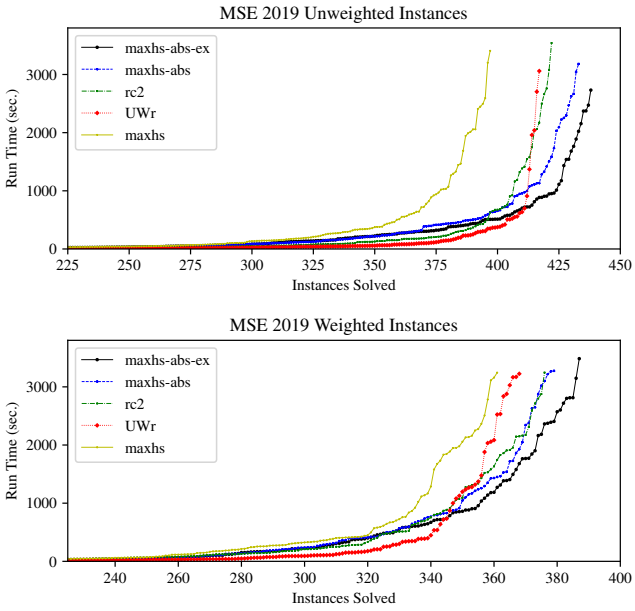

Figure 1: Cactus plot of solver performance on the 599 unweighted (top) and 586 weighted (bottom) instances of MSE 2019.

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 demosntrated 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

[Andres *et al.*, 2012] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of ICLP*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[Ansótegui *et al.*, 2013] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[Asín *et al.*, 2009] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In Oliver Kullmann, editor, *Proc SAT*, volume 5584 of *LNCS*, pages 167–180. Springer, 2009.

[Bacchus *et al.*, 2017] Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in maxsat. In *Proc. CP*, volume 10416 of *LNCS*, pages 641–651. Springer, 2017.

[Bacchus *et al.*, 2019a] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maxsat evaluation 2018: New developments and detailed results. *J. Satisf. Boolean Model. Comput.*, 11(1):99–131, 2019.

[Bacchus *et al.*, 2019b] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins, editors. *MaxSAT Evaluation 2019: Solver and Benchmark Descriptions*. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, Finland, 2019.

[Bacchus *et al.*, 2021] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *Maximum Satisfiability*, chapter 24, pages 929–991. Frontiers in Artificial Intelligence and Applications. IOS Press BV, 2021.

[Bailleux and Boufkhad, 2003] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Proc CP*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003.

[Berg *et al.*, 2020] Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set maxsat solving. In *SAT*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020.

[Blondel *et al.*, 2008] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, Oct 2008.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Proc CP*, volume 6876 of *LNCS*, pages 225–239. Springer, 2011.

[Davies and Bacchus, 2013a] Jessica Davies and Fahiem Bacchus. Exploiting the power of mip solvers in maxsat. In Matti Järvisalo and Allen Van Gelder, editors, *Proc SAT*, volume 7962 of *LNCS*, pages 166–181. Springer, 2013.

[Davies and Bacchus, 2013b] Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian Schulte, editor, *Proc CP*, volume 8124 of *LNCS*, pages 247–262. Springer, 2013.

[Davies, 2013] Jessica Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. PhD thesis, University of Toronto, 2013.

[Eén and Sörensson, 2003] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electron. Notes Theor. Comput. Sci.*, 89(4):543–560, 2003.

[Ignatiev *et al.*, 2019] Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019.

[Karpinski and Piotrów, 2019] Michal Karpinski and Marek Piotrów. Encoding cardinality constraints using multiway merge selection networks. *Constraints*, 24(3-4):234–251, 2019.

[Koshimura *et al.*, 2012] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. Qmaxsat: A partial max-sat solver. *J. Satisf. Boolean Model. Comput.*, 8(1/2):95–100, 2012.

[Morgado *et al.*, 2014] António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided maxsat with soft cardinality constraints. In Barry O'Sullivan, editor, *Proc CP*, volume 8656 of *LNCS*, pages 564–573. Springer, 2014.

[Ogawa *et al.*, 2013] Toru Ogawa, Yangyang Liu, Ryuzo Hasegawa, Miyuki Koshimura, and Hiroshi Fujita. Modulo based CNF encoding of cardinality constraints and its application to maxsat solvers. In *Proc ICTAI*, pages 9–17. IEEE Computer Society, 2013.

[Saikko *et al.*, 2016] Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid maxsat solver. In Nadia Creignou and Daniel Le Berre, editors, *Proc SAT*, volume 9710 of *LNCS*, pages 539–546. Springer, 2016.

[Saikko, 2015] Paul Saikko. Re-implementing and extending a hybrid SAT-IP approach to maximum satisfiability. Master's thesis, University of Helsinki, 2015.

[Sinz, 2005] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Proc CP*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.