Towards the unification of the Core-Guided and Hitting Set Maximum Satisfiability approaches

Imko Marijnissen

Delft University of Technology

TU Delft

Towards the unification of the Core-Guided and Hitting Set Maximum Satisfiability approaches

by

Imko Marijnissen

Thesis Committee

Dr. E. Demirović	TU Delft
Dr. S. Picek	TU Delft
M. Flippo	TU Delft

The thesis is part of the project "Towards a Unification of AI-Based Solving Paradigms for Combinatorial Optimisation" (OCENW.M.21.078) of the research programme "Open Competition Domain Science - M" which is financed by the Dutch Research Council (NWO).

Acknowledgements

Looking back on the journey of writing this thesis, it has been a story of ups and downs which I have come to learn is a part of the vicissitudes of research. I have loved the field of maximum satisfiability since I initially learned of it; it has been a joy to be able to further familiarize myself with this research area and, hopefully, to contribute a bit to advance it.

I would like to thank my supervisors Emir Demirović and Maarten Flippo for all of their feedback, discussions and advice, without which this thesis would not have been what it is today. I would also like to thank Ana Tatabitovska for all the fruitful conversations and support. Finally, I would like to thank Nina Narodystka for not only providing the core data which allowed for a more granular analysis but also the enlightening discussions.

All of this would not have been possible without the support of friends and family who have inspired me to persevere for as long as I can remember. I would especially like to thank my mother and sibling for always being a grounding voice of reason in the tumultuousness of it all.

Imko Marijnissen
August 2023

Abstract

Core-guided solvers and Implicit Hitting Set (IHS) solvers have become ubiquitous within the field of Maximum Satisfiability (MaxSAT). While both types of solvers iteratively increase the solution cost until a satisfiable solution is found, the manner in which this relaxation is performed leads to the belief that these techniques are wholly separate from one another. This work exhibits the difficulty of directly translating the cores found by an IHS-solver to cores utilizable by an OLL-solver due to an inherent disconnect between the manner in which both approaches explore the solution space. It will then be shown how this translation can be performed more easily given certain assumptions. Finally, several techniques are introduced for performing a translation from cores of the original formula to OLL-cores which avoids the aforementioned issue, resulting in a hybrid IHS-OLL algorithm. The comparison between the performance of the hybrid algorithm and `RC2` shows that the hybrid algorithm achieves analogous performance in terms of the number of instances solved while indicating that utilising cores of the original formula as a starting point for an OLL solver can be beneficial for the performance of the solver in certain cases.

Contents

1 Introduction

The field of Maximum Satisfiability (MaxSAT) is ubiquitous due to the fact that numerous practical problems, such as planning [1], clustering [2], explainable machine learning [3] and bioinformatics [4] can be described as a MaxSAT problem instance. A MaxSAT instance consists of propositional logic formulas in conjunctive normal form and solvers attempt to find a solution which satisfies the formula in addition to an optimization criteria. More formally, MaxSAT algorithms take as input two sets of clauses, the hard clauses and the soft clauses, and produce an assignment of literals such that it satisfies all of the hard clauses and minimizes the number of falsified soft clauses. The solving techniques employed by MaxSAT solvers have tremendously increased in performance in recent times. An overview of the primary strategies employed within this field is visualized in Figure 1.1 which exemplifies the variety of solvers.

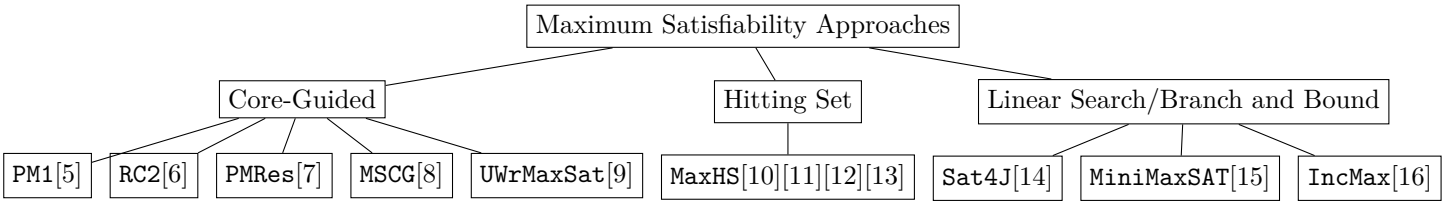


Figure 1.1: Overview of Maximum Satisfiability Approaches

Recent research has primarily focused on two branches of approaches, namely the core-guided and (implicit) hitting set (IHS) algorithms. Both approaches make use of so-called cores, a set of soft clauses such that together with the hard clauses it forms an unsatisfiable instance. The core-guided algorithms iteratively relax these cores and employ cardinality constraints to transform the working formula. The implicit hitting set algorithms attempt to prevent the working formula from becoming exceedingly complex due to the addition of constraints by employing a minimum hitting set sub-routine. Both techniques have shown laudable performance in contemporary competitions illustrating their applicability on a variety of instances.

Recent work [17] has demonstrated that there exists a connection between the cores found by `PM1` and `MaxHS` which specifies that the cores found by both algorithms possess similar properties. However, the `PM1` algorithm and its derivatives (such as `PMRes`) have predominantly been replaced by solvers making use of the so-called OLL algorithm [18] such as `RC2`, `MSCG` and `UWrMaxSAT`. This work addresses the possibility of a (similar) connection existing between the cores found by the IHS solvers and the cores found by the OLL solvers.

To this end, this work will show the difficulty of simulating OLL based on IHS solely based on the information provided by an IHS-solver. It is then demonstrated that this simulation can be performed under certain assumptions. This reasoning is further concretized by proposing an algorithm which performs this simulation by creating structures with a certain property. Moreover, a practical hybrid algorithm is proposed which makes use of an IHS-solver to generate cores for a limited duration after which this information is translated to an OLL-solver which completes the

solving process. Two techniques are proposed which perform this translation while circumventing the aforementioned simulation issues. The experimentation which compares the hybrid algorithm to the OLL-solver `RC2` shows that, on certain instances, the performance is significantly improved. In terms of the number of instances solved, particular combinations of techniques outperform `RC2` by some margin which indicates that there is potential for making use of both IHS and OLL to solve a MaxSAT instance efficiently. Additionally, the usage of IHS with non-optimal hitting sets further improves the overall performance of the hybrid algorithm for certain techniques.

The remainder of the work is structured as follows. Section 2 discusses preliminaries and definitions which are used throughout the rest of the work. Section 3 exhibits research related to the connection between core-guided algorithms and the IHS algorithm which have served as a source of inspiration. Section 4 introduces a theoretical basis as to why directly translating from cores found by an IHS-solver to OLL-cores is strenuous followed by a description of the assumption required for allowing such a simulation in a straightforward manner. Moreover, this section describes the property required of a simulated OLL-core and provides an outline of a potential algorithm for performing a direct translation under the assumption. Section 5 specifies the general pipeline for a hybrid IHS-OLL algorithm while avoiding the issues mentioned in the previous section. Furthermore, this section describes the techniques which are proposed in this work for translating from cores of the original formula to OLL-cores and certain practical considerations involved. Finally, Section 6.3 provides an overview of the performance of the different techniques employed by the hybrid algorithm on approximately 600 unweighted instances and compares these results to the performance of the OLL-solver `RC2`. Moreover, this section analyses the discrepancies between the instances solved by the different techniques and proposes/evaluates a portfolio strategy based on a decision tree.

2

Preliminaries

2.1. SAT

The satisfiability (SAT) problem, serves as the foundation of the MaxSAT problem and can be encountered in many real-life applications such as hardware verification [19], planning problems [20] and bounded model checking [21]. The problem consists of a set of clauses F where a clause is a disjunction of boolean literals (e.g. $C_k = x_1 \vee ... \vee x_n$ and $x_i \in \{0, 1\}$). The negation of a boolean literal x_i is denoted by $\bar{x}_i$. This work will treat each clause as a set rather than a disjunction of literals (since duplicate literals do not influence the satisfiability of a clause). The set of all boolean literals in F is denoted by $vars(F)$.

A SAT-solver produces an assignment I such that every variable in $vars(F)$ is mapped to a 0 or a 1, i.e. $I : vars(F) \mapsto \{0, 1\}$. A clause C_k is satisfied if $\exists x_i \in C_k$ s.t. $I(x_i) = 1$, this is denoted by $I(C_k) = 1$. A clause C_k is unsatisfied if $\forall x_i \in C_k$ s.t. $I(x_i) = 0$, this is denoted by $I(C_k) = 0$. A formula F (consisting of a set of clauses) is said to be unsatisfiable if $\nexists I$ s.t. $\forall C_k \in F : I(C_k) = 1$ and satisfiable otherwise. The goal of a SAT-solver is thus to produce an assignment such that all of the clauses in F are satisfied. An example of a satisfiable formula can be seen in Example 1 and an example of an unsatisfiable formula can be seen in Example 2.

Example 1
Let's say that we have a formula $F = \{\{x_1, x_2, x_3\}, \{\bar{x}_1, \bar{x}_2\}\}$. It can be seen from this formula that there exist multiple assignments I such that all of the clauses are satisfied, an example of such an assignment being $I = \{x_1 = 1, x_2 = 0, x_3 = 1\}$

Example 2
Let's say that we have a formula $F = \{\{x_1, x_2\}, \{\bar{x}_1, x_2\}, \{x_1, \bar{x}_2\}, \{\bar{x}_1, \bar{x}_2\}\}$. It can be seen from this formula that there exists no assignment I for which all of the clauses are satisfied. It can thus be said that F is unsatisfiable.

2.1.1. SAT solving

This work makes use of a SAT-solver called `Glucose` [22]. This solver implements the so-called CDCL (conflict-driven clause learning) algorithm consisting of a search algorithm with backtracking in combination with dynamically learning clauses from conflicts (called learnt clauses) to prevent unnecessary exploration of the solution space. Besides solving the SAT problem in an efficient manner, modern SAT-solvers extract a set of clauses called a core which can explain the unsatisfiability of the provided formula. These cores are oftentimes utilised by MaxSAT algorithms.

An incremental SAT-solver is a SAT-solver which is called multiple times with similar formulas and which is optimized for such usage. The number of SAT-solvers which are incremental SAT-solvers has increased drastically in recent years due to its wide applicability within the context of MaxSAT. A number of aspects related to the search process are influenced by the fact that

numerous calls to the SAT-solver are made rather than a single one (e.g. heuristics, learnt clause addition/deletion).

In addition to being incremental, a variety of SAT-solvers make use of the assumption interface (an efficient version of which was introduced in [23]). Assumptions allow the user to provide a set of literals such that these literals are necessarily picked as decision literals at the root level during the search process of the solver. There are numerous benefits to this scheme, for example, the learnt clauses which are generated are independent of the assumptions meaning that the learnt clauses are valid between invocations of the SAT-solver. Furthermore, the cores which are generated by a SAT-solver utilising the assumption interface consist solely of assumption literals (the usefulness of this technique in the context of MaxSAT solving will be exhibited in the following sections).

In general, the field of satisfiability solving is active due to its ubiquitous nature. To this end, a competition[1] is organised which compares a wide variety of solvers on numerous benchmarks.

2.2. MaxSAT

The maximum satisfiability (MaxSAT) problem, is closely related to the aforementioned SAT problem. As opposed to the SAT problem, the MaxSAT problem consists of two sets of clauses, the hard clauses F_h and the soft clauses F_s (which are assumed to be unit clauses in this work) where $F = F_h \cup F_s$. The goal of the (unweighted) maximum satisfiability algorithm is to find an assignment I such that all hard clauses are satisfied and the minimum number of soft clauses are unsatisfied, more formally, it produces an assignment I s.t. $\forall C_k \in F_h : I(C_k) = 1$ and $|\{C_j \in F_s \mid I(C_j) = 0\}|$ is minimal. Thus, the MaxSAT problem can be interpreted as the optimization version of the SAT problem, in which the hard clauses form a "regular" SAT instance and the set of soft clauses introduce the optimization component. An example of a MaxSAT instance can be seen in Example 3.

Example 3
Let's say we have a formula consisting of the hard clauses $F_h = \{\{x_1, x_2\}, \{x_3, x_4\}\}$ and the soft clauses $F_s = \{\{\bar{x}_1\}, \{\bar{x}_2\}, \{\bar{x}_3\}, \{\bar{x}_4\}\}$. There are numerous solutions to this particular formula, for example, the assignment $I_1 = \{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0\}$ which has a solution cost of three. The optimal solution cost of F is two, indicating that at least two of the soft clauses are violated, an example of an optimal solution is $I_2 = \{x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0\}$.

While this work focuses on unweighted partial MaxSAT (in which each soft clause has a weight of one), it should be noted that there are more versions such as weighted MaxSAT (in which each soft clause has a weight which is possibly different than one), incomplete MaxSAT[2] (in which the aim of a solver is to find the best possible solution in a limited time) and incremental MaxSAT[3] (in which a solver is tasked with solving a sequence of related instances).

2.2.1. MaxSAT solving

This section will introduce the approaches related to MaxSAT solving which are utilised within this work.

Core-guided MaxSAT

Definition 1
A **core** is a subset of clauses $\kappa \subseteq F_s$ such that $F_h \cup \kappa$ is unsatisfiable. A core is said to be minimal (or MUS) if $\nexists C_k \in \kappa$ s.t. $F_h \cup (\kappa \setminus \{C_k\})$ is unsatisfiable, i.e. a core is minimal if no proper subset is a core.

Definition 2
A **cardinality constraint** is a constraint that restricts the number of elements allowed to be

[1] https://satcompetition.github.io/2023/
[2] https://maxsat-evaluations.github.io/2022/rules.html#:~:text=Main%20tracks%20for%20incomplete%20solvers
[3] https://maxsat-evaluations.github.io/2022/incremental.html

set to true within the set of variables participating in a core. Let $\mathcal{V}$ be a set of variables (e.g. a clause), y be an integer such that $0 \leq y \leq |\mathcal{V}|$ and $\bigotimes \in \{<, \leq, =, \geq, >\}$ then a cardinality constraint is of the form $\sum_{x_i \in \mathcal{V}} x_i \bigotimes y$

These cardinality constraints cannot be used in SAT/MaxSAT directly as SAT-solvers operate on clauses rather than cardinality constraints. A popular encoding which encodes a cardinality constraint into clauses is presented in [24]. In this work, it is assumed that each cardinality constraint is represented by a single boolean literal.

The initial core-guided algorithm `PM1` was proposed by Fu and Malik in [5], paving the way for core-guided algorithms to dominate the MaxSAT field due to the improved guidance as compared to linear search [14], [25]. The `PM1` algorithm consists of a call to a SAT-solver such as `Glucose` [22], `Cadical`[26] or `MiniSAT` [27] and extracting a core (see Definition 1) of the formula. For each soft clause in the core, a so-called relaxation variable (also known as a blocking variable) is appended to the clause and a cardinality constraint (see Definition 2) is added to F_h which allows a single of these relaxation variables to be set to true. Practically, allowing a relaxation variable to be set to true means that a soft clause can be falsified. This procedure continues until a satisfiable formula is found. The pseudo-code of this approach can be seen in Algorithm 1.

Algorithm 1 `PM1`[5]

Input: F - The Formula
Output: An Optimal Cost and Assignment

```
1:  procedure PM1
2:      i ← 0
3:      cards ← {}
4:      while True do
5:          (SAT, κ, I) ← SATSolve(F_h ∪ F_s ∪ cards)
6:          if SAT then
7:              return i, I
8:          i ← i + 1
9:          R ← {}
10:         for C_k ∈ κ do
11:             F_s ← (F_s \ C_k) ∪ {C_k ∪ r_k^i}
12:             R ← R ∪ {r_k^i}
13:         cards ← cards ∪ (Σ_{r_k^i ∈ R} r_k^i = 1)
```

The algorithm starts by performing a call to a SAT-solver (Line 5) with the set of hard clauses, soft clauses and cardinality constraints. This call returns three results:

1. Whether the provided formula is satisfiable (stored in SAT)
2. If the formula is unsatisfiable then κ will contain a core of the formula
3. Finally, if the formula is satisfiable then the SAT-solver will return an assignment of variables I

If the formula is satisfiable then the optimal solution cost and its corresponding assignment can be returned (Line 7). If this is not the case then the cost is increased by one and all of the clauses in κ are relaxed using a fresh relaxation variable r_k^i. The original clause is then removed from the formula and the relaxed clause is added. Finally, a cardinality constraint specifying that a single of these relaxation variables can be set to true is added (Line 13). This process is iterated until enough soft clauses have been relaxed such that a satisfiable formula has been created. Example 4 shows how the algorithm works on a small example. It should be noted that this work focuses on the unweighted MaxSAT problem, in this context the cost is increased by one in each iteration of the `PM1`, however, this is not necessarily the case for weighted MaxSAT. An algorithm for weighted MaxSAT (`WPM1`) based on `PM1` is proposed in [28].

Example 4
Let's say we have a formula consisting of the hard clauses $F_h = \{\{x_1, x_2\}, \{x_3, x_4\}\}$ and the soft clauses $F_s = \{\{\bar{x}_1\}, \{\bar{x}_2\}, \{\bar{x}_3\}, \{\bar{x}_4\}\}$. Table 2.1 shows an execution trace when performing Algorithm 1 on the aforementioned formula.

$\mathbf{i}$	$\kappa^{\mathbf{i}}$	$\mathbf{cards^i}$	$\mathbf{F_s^i}$
0	$\{\bar{x}_1, \bar{x}_2\}$	$\{r_1^1 + r_2^1 = 1\}$	$\{\{\bar{x}_1, r_1^1\}, \{\bar{x}_2, r_2^1\}, \{\bar{x}_3\}, \{\bar{x}_4\}\}$
1	$\{\bar{x}_3, \bar{x}_4\}$	$\{r_1^1 + r_2^1 = 1,\ r_3^2 + r_4^2 = 1\}$	$\{\{\bar{x}_1, r_1^1\}, \{\bar{x}_2, r_2^1\}, \{\bar{x}_3, r_3^2\}, \{\bar{x}_4, r_4^2\}\}$
2	$\{\}$	-	-

Table 2.1: Execution trace of `PM1`

Iteration 0
The SAT-solver is called with the formula $F_h \cup F_s$ which will return unsatisfiable. The corresponding core $\kappa^0 = \{\bar{x}_1, \bar{x}_2\}$ is extracted (such that $F_h \cup \kappa^0$ is not satisfiable). The soft clauses are transformed into $F_s = \{\{\bar{x}_1, r_1^1\}, \{\bar{x}_2, r_2^1\}, \{\bar{x}_3\}, \{\bar{x}_4\}\}$ and $cards^0 = \{r_1^1 + r_2^1 = 1\}$.

Iteration 1
The SAT-solver is called with the adjusted formula which will return unsatisfiable. The core $\kappa^1 = \{\bar{x}_3, \bar{x}_4\}$ is then extracted. The soft clauses are transformed into $F_s = \{\{\bar{x}_1, r_1^1\}, \{\bar{x}_2, r_2^1\}, \{\bar{x}_3, r_3^2\}, \{\bar{x}_4, r_4^2\}\}$ and $cards^1 = \{r_1^1 + r_2^1 = 1,\ r_3^2 + r_4^2 = 1\}$.

Iteration 2
The SAT-solver is called with the adjusted formula and will return satisfiable. This indicates that the optimal solution has a cost of two (as i = 2).

The main drawback of this algorithm lies in the fact that the introduction of the cardinality constraints and the relaxation variables oftentimes induce additional strain on the SAT-solver compared to methods which do not utilise cardinality constraints to solve the MaxSAT problem. This can lead to decreased solving time in consecutive iterations.

It should be noted that the call to the SAT-solver is oftentimes performed using the assumption interface where the soft clauses (which are assumed to be unit) are provided as the assumptions. This ensures that κ solely contains soft clauses.

OLL

The work by Fu and Malik originated another approach which is part of the core-guided family, namely the OLL algorithm based on [18]. Similar to `PM1`, this approach likewise utilises relaxation variables and cardinality constraints to transform the working formula. However, in contrast to `PM1` which solely allows clauses of the original formula to be part of cores, OLL creates cores consisting of both clauses of the original formula **and** the created cardinality constraints. Whenever a core is found by the SAT-solver, an OLL-solver relaxes the clauses in this core with relaxation variables (or increases the right-hand side by one in case of cardinality constraints) and creates a cardinality constraint over the relaxation variable(s)/cardinality constraint(s) specifying that an additional variable can become true in a structured manner. As opposed to `PM1`, it then removes the relaxed clauses of the original formula from the soft clauses and appends them to the hard clauses. Additionally, it adds the created cardinality constraint to the soft clauses such that this constraint can become part of subsequent cores. This robust approach relying on the expressivity of cardinality constraints was shown to perform well in comparison to prior approaches. The pseudo-code of a solver (`RC2`) which implements the OLL algorithm can be seen in Algorithm 2.

Algorithm 2 RC2[6]

Input: F - The Formula
Output: An Optimal Cost and Assignment

```
1:  procedure RC2
2:      i ← 0
3:      while True do
4:          L = ∅
5:          (SAT, κ, I) ← SATSolve(F)
6:          if SAT == True then
7:              return i, I
8:          else
9:              i ← i + 1
10:             for C_k ∈ κ do
11:                 if C_k is not a cardinality constraint then
12:                     F_s ← F_s \ C_k
13:                     F_h ← F_h ∪ (C_k ∪ r_i)
14:                     L ← L ∪ {r_i}
15:                 else
16:                     L ← L ∪ {C_k}
17:                     rhs(C_k) ← rhs(C_k) + 1
18:             F_s ← F_s ∪ (∑_{x_i∈L} x_i ≤ 1)
```

It starts by performing a call to a SAT-solver to check for the satisfiability of $F = F_h \cup F_s$. In case this formula is satisfiable, it returns the solution cost (i.e. the number of relaxation steps performed) and the corresponding assignment. Otherwise, it goes through all of the clauses present in the core. If a clause in the core is a soft clause (Line 11) then this clause is relaxed with a fresh relaxation variable, removed from the soft clauses and subsequently added to the hard clauses. If a clause in the core is a cardinality constraint (Line 15), which is represented by a single boolean literal as per Definition 2, then the right-hand side of this cardinality constraint is increased by one if possible. Finally, it creates a new cardinality constraint stating that one more variable in the soft clauses and/or cardinality constraint can be set to true (Line 18). Example 5 shows the working of the algorithm.

Example 5
Let's say we have a formula consisting of the hard clauses $F_h = \{\{x_1, x_2\}, \{x_3, x_4\}\}$ and the soft clauses $F_s = \{\{\bar{x}_1\}, \{\bar{x}_2\}, \{\bar{x}_3\}, \{\bar{x}_4\}\}$. Table 2.2 shows an execution trace when performing Algorithm 2 on the aforementioned formula.

$\mathbf{i}$	$\kappa^{\mathbf{i}}$	$\mathbf{F_s^i}$	$\mathbf{F_h^i}$
0	$\{\bar{x}_1, \bar{x}_2\}$	$\{r_1 + r_2 \leq 1, \{\bar{x}_3\}, \{\bar{x}_4\}\}$	$\{\{x_1, x_2\}, \{x_3, x_4\}, \{\bar{x}_1, r_1\}, \{\bar{x}_2, r_2\}\}$
1	$\{r_1 + r_2 \leq 1, \bar{x}_3, \bar{x}_4\}$	$\{r_1 + r_2 \leq 2, r_3 + r_4 + (r_1 + r_2 > 1) \leq 1\}$	$\{\{x_1, x_2\}, \{x_3, x_4\}, \{\bar{x}_1, r_1\}, \{\bar{x}_2, r_2\}, \{\bar{x}_3, r_3\}, \{\bar{x}_4, r_4\}\}$
2	$\{\}$	-	-

Table 2.2: Execution trace of OLL

Iteration 0
The SAT-solver is called with $F = F_h \cup F_s$ and it will return unsatisfiable. The core $\kappa^0 = \{\{\bar{x}_1\}, \{\bar{x}_2\}\}$ is then extracted. All of the soft clauses in κ^0 are relaxed and added to the hard clauses resulting in $F_h^0 = \{\{x_1, x_2\}, \{x_3, x_4\}, \{\bar{x}_1, r_1\}, \{\bar{x}_2, r_2\}\}$ after which the cardinality constraint is added to the soft clauses resulting in $F_s^0 = \{r_1 + r_2 \leq 1, \{\bar{x}_3\}, \{\bar{x}_4\}\}$.

Iteration 1
The SAT-solver is called with the updated formula and it returns unsatisfiable. It then returns the core $\kappa^1 = \{r_1 + r_2 \leq 1,\ \bar{x}_3,\ \bar{x}_4\}$ (note that this is not a MUS). In this case, the right-hand side of the cardinality constraint in the core is increased and a new cardinality constraint stating that one more variable can be set to true is created.

Iteration 2
The SAT-solver is called with the updated formula and it returns satisfiable. This means that an optimal solution has been found of size two.

As mentioned previously, the main difference between `PM1` and `RC2` is that it is possible for the cores found by an OLL-solver to contain cardinality constraints. Such a core which can contain both cardinality constraints and soft clauses is called an OLL-core (see Definition 3). This limits the number of cardinality constraints which are required to be encoded by the solver by reusing cardinality constraints over a set of variables (e.g. if there is a set of variables $\mathcal{V}$ and two variables are required to be set to true in any solution then `PM1` would create two separate cardinality constraints while an OLL-solver would create a single cardinality constraint). However, the overhead required for encoding these cardinality constraints into clauses is still present (though to a lesser extent) which exhibits the main drawback of this method.

Definition 3
A core which can contain both soft clauses of the original formula **and** cardinality constraints is called an **OLL-core**. An example of an OLL-core could be $\kappa = \{x_1, x_2, r_3 + r_4 \leq 1\}$.

Currently, a majority of the state-of-the-art solvers participating in the MaxSAT evaluation 2022[4] make use of the OLL-algorithm in combination with other techniques such as:

1. **Weight-aware core extraction [29]:** Applicable primarily to weighted MaxSAT instances, this technique aims to extract a multitude of cores of the working formula before transforming the formula, i.e. adding a cardinality constraint. In the unweighted case, this technique is referred to as the disjoint core technique in which disjoint cores are extracted before reformulation.
2. **Intrinsic atmost1 Detection [6]:** This technique makes use of the fact that if a certain set of clauses $S \subset F_s$ is present in the formula such that $\sum_{x_i \in S} x_i \leq 1$ then the solution cost is required to be at least $|S| - 1$ and a simplification of the instance can occur by replacing S. These (sub)sets of soft clauses can be detected by making use of propagation and checking whether other literals in the set are implied by this propagation.
3. **Core Minimization [30]:** These techniques reduce the size of the cores found by the SAT-solver to curtail the magnitude of the produced cardinality constraints and to prevent the solver from considering whether a solution contains literals which are not part of the current core.

Implicit Hitting Sets

Definition 4
A **hitting set** in the context of the implicit hitting set approach for MaxSAT is a subset of clauses $H \subseteq F_s$ over the set of cores $\mathcal{K}$ such that $\forall \kappa_i \in \mathcal{K} : \kappa_i \cap H \neq \emptyset$. A minimum hitting set is a hitting set such that there does not exist another H' such that $|H'| < |H|$.

Definition 5
A **correction set** is a set of clauses $cs \subseteq F_s$ such that $F_h \cup (F_s \setminus cs)$ is satisfiable. A minimum correction set is a correction set such that there does not exist another cs' such that $|cs'| < |cs|$.

[4] https://maxsat-evaluations.github.io/2022/

Certain limitations of PM1 were addressed in [10] which introduces the implicit hitting set approach for solving MaxSAT. This approach allows for the simplification of the problems solved by the SAT-solver by making use of an NP-hard minimum hitting set (see Definition 4) sub-routine (using a solver such as CPLEX[5]) as opposed to the increased strain on the SAT-solver induced by the added cardinality constraints of PM1. The solver retains a set of cores and calculates a minimum hitting set over these cores which corresponds with a potential minimum correction set (see Definition 5). It iterates this process, finding cores to reject all potential minimum hitting sets of size y after which it raises the bound, indicating that a minimum correction set is at least of size $y+1$. If a minimum hitting set is found to be a correction set (i.e. $F_h \cup (F_s \setminus hs)$ is satisfiable) then an optimal solution has been found. The pseudo-code of an IHS-solver (namely MaxHS) can be seen in Algorithm 3.

Definition 6
MinHS($\mathcal{K}$) returns a minimum hitting set of the set $\mathcal{K}$

Algorithm 3 MaxHS [10]

Input: F - The Formula
Output: An Optimal Cost and Assignment

```
1: procedure MaxHS
2:     K ← ∅
3:     while True do
4:         hs ← MinHS(K)
5:         (SAT, κ) ← SATSolve(F \ hs)
6:         if SAT then
7:             return |hs|, hs
8:         else
9:             K ← K ∪ κ
```

The algorithm starts by calculating a minimum hitting set over the current set of cores $\mathcal{K}$. It then removes the clauses present in this minimum hitting set from the formula and checks for satisfiability. In case of satisfiability, the optimal solution has been found and the cost is equal to the size of the minimum hitting set. If the SAT-call returns that the formula is unsatisfiable then the new core κ which rejects the current minimum hitting set is added to the set of cores $\mathcal{K}$. This process iterates until the minimum hitting set is found to be a minimum correction set. Example 6 shows the workings of the algorithm:

Example 6
Let's say we have a formula consisting of the hard clauses $F_h = \{\{x_1, x_2\}, \{x_3, x_4\}\}$ and the soft clauses $F_s = \{\{\bar{x}_1\}, \{\bar{x}_2\}, \{\bar{x}_3\}, \{\bar{x}_4\}\}$. Table 2.3 shows an execution trace when performing Algorithm 3 on the aforementioned formula.

i	$\mathcal{K}$	hs^{i}	κ^{i}
0	$\{\}$	$\{\}$	$\{\bar{x}_1, \bar{x}_2\}$
1	$\{\{\bar{x}_1, \bar{x}_2\}\}$	$\{\bar{x}_1\}$	$\{\bar{x}_3, \bar{x}_4\}$
2	$\{\{\bar{x}_1, \bar{x}_2\}, \{\bar{x}_3, \bar{x}_4\}\}$	$\{\bar{x}_1, \bar{x}_3\}$	$\{\}$

Table 2.3: Execution trace of IHS

Iteration 0
The SAT-solver is called with the set of clauses $F = F_h \cup F_s$ and it will return unsatisfiable. The core $\kappa^0 = \{\bar{x}_1, \bar{x}_2\}$ is extracted. This core is added to the current set of cores resulting

[5]https://www.ibm.com/docs/en/icos/12.8.0.0?topic=cplex

in $\mathcal{K} = \{\{\bar{x}_1, \bar{x}_2\}\}$.

Iteration 1
A new minimum hitting set of $\mathcal{K}$ is then calculated, resulting in $hs^1 = \{\bar{x}_1\}$. The SAT-solver is then called with the formula $F_h \cup (F_s \setminus hs^1)$ which results in the SAT-solver returning unsatisfiable. A new core $\kappa^1 = \{\bar{x}_3, \bar{x}_4\}$ is then extracted. This core is then added to the current set of cores resulting in $\mathcal{K} = \{\{\bar{x}_1, \bar{x}_2\}, \{\bar{x}_3, \bar{x}_4\}\}$.

Iteration 2
A new minimum hitting set of $\mathcal{K}$ is then calculated, resulting in $hs^2 = \{\bar{x}_1, \bar{x}_3\}$. The SAT-solver is then called with the formula $F_h \cup (F_s \setminus hs^2)$ which results in the SAT-solver returning satisfiable. This means that an optimal solution has been found of size two.

The main drawback of this algorithm is the overhead required for calculating a minimum hitting set. The time spent within the minimum hitting set solver is time which cannot be spent solving a SAT instance. There is thus a tradeoff between the time spent on calculating a minimum hitting set and the time spent in the SAT-solver. Results show that this time spent within the minimum hitting set solver is compensated for by the reduced amount of time required to be spent in the SAT-solver.

`MaxHS` has been enhanced with multiple adjustments to improve upon its initial version allowing it to compete with the current state-of-the-art OLL-solvers:

1. **Postponing exact calls [11]:** One of the principal bottle-necks of `MaxHS` is the expensive exact calls to the minimum hitting set solver. To avoid this issue, non-optimal hitting sets can be computed in combination with a lower/upper-bounding technique to decrease the number of required exact calls made to the minimum hitting set solver.
2. **Abstract cores [12]:** An additional impediment of the implicit hitting set technique employed by `MaxHS` is that, on certain instances, it is required to extract copious amounts of cores to prove optimality. To this end, cardinality constraints, similar to the ones used by OLL-solvers, are used over a subset of variables to represent numerous cores using a single constraint.
3. **Reduced cost fixing [13]:** This technique utilizes a well-known approach from integer programming (IP) to fix the values of certain variables based on the lower and upper bound of an instance. This technique is principally useful in the context of weighted MaxSAT instances due to the nature of the procedure.

Certain optimizations discussed in the prior paragraphs have been utilised within this work or have served as an inspiration for the design choices of the proposed algorithms.

3

Related Work

This section discusses work related to evincing the connection between the aforementioned MaxSAT techniques and their findings.

Firstly, [31] shows that there exists a relation between the cores found by `PM1` and the cores of the original formula. The cores found by `PM1` are not cores of the original formula but rather they are cores of the current working formula F^i which includes the cardinality constraints and the relaxation variables. It is shown that the cores found by `PM1` are composed of a union of cores of the original formula. This provides the initial evidence of a connection between the algorithms of `PM1` and IHS (as the initial version of IHS solely extracts cores of the original formula).

This potential connection is further explored by [17] which shows that the set of cores generated by both algorithms boast similar properties concerning the size of the minimum hitting set and how these hitting sets relate to the generated lower bounds. Their findings are summarized by the following three properties, where each algorithm has generated a set of cores $S = [\kappa^0, ..., \kappa^{opt}]$ and a lower bound lb^i for a solution with optimal cost opt:

1. $S[opt] \subset$ All-MUS(F_s), stating that there exists a formula for which both algorithms do not find all MUSes of the formula (see Definition 7 for the definition of All-MUS).
2. |MinHS(S[i])| $\geq lb^i$, stating that the minimum hitting set of the cores found up and until iteration i, is at least as large as the lower-bound found by the algorithm.
3. |MinHS($S[opt]$)| = opt, stating that the size of the minimum hitting set of all cores found by either algorithm is equal to the optimal solution cost (as this minimum hitting set corresponds with a minimum correction set of the formula).

Definition 7
All-MUS(**F**) returns all possible MUSes (see Definition 1) of the formula F.

Finally, [32] has explored the connection between combining different maximum satisfiability approaches. In [32], the linear search algorithm (which creates a single cardinality constraint whose right-hand side is adjusted iteratively to find the optimal solution) is combined with core-guided algorithms in an attempt to improve performance. It should be noted that [32] focuses on incomplete MaxSAT in which the goal is not necessarily to find the optimal solution but rather to find a "good" solution in a short timeframe.

The so-called core-boosted linear search makes use of two phases, the core-guided phase and the linear phase. First of all, the core-guided phase executes a core-guided solver given certain resource constraints. If it finds the optimal solution then this solution is returned. If it did not solve to optimality then the final working formula F^i is returned and its best-found solution I^*. The linear phase then makes use of the final working formula found in the previous phase and I^* is utilised as the initial solution allowing the linear search algorithm to infer an upper bound on the solution cost. This phase is then run until either an optimal solution has been found or until the available resources have been depleted.

[32] has served as an inspiration for the hybrid algorithm proposed in this work.

4

Simulation

This section specifies the limitations related to simulating the OLL algorithm based on the cores found by an IHS-solver. Furthermore, it describes the assumptions required for overcoming these limitations and an outline of an algorithm for performing the simulation based on the aforementioned assumptions.

4.1. Simulating OLL based on IHS

Definition 8
A **simulation** of OLL based on IHS is defined as follows: Given a set of cores generated by an IHS-solver, translate these cores to OLL-cores without making use of a SAT-oracle such that an OLL-solver provided with these OLL-cores can solve to optimality.

The goal of this simulation is thus to find OLL-cores solely based on the information discovered by an IHS-solver. The simulation requires that the optimal solution can be found using a (partially) simulated set of cores by an OLL-solver, i.e. an OLL-solver would be able to find the optimal solution using the simulated cores. The simulation does not allow any calls to a SAT-solver but it does permit the usage of supplementary superpolynomial procedures to create the cores (e.g. a minimum hitting set procedure).

Based on Definition 8, it should be determined when such a translation could take place. As stated in Section 2.2.1.2, OLL-solvers raise the lower bound on the optimal solution cost by one every time that a core is found which indicates a natural translation point. Intuitively, if the bound is raised by the IHS-solver then this would correspond with a point in the solving process at which the translation can be performed to an OLL-core as, similarly to an iteration of the OLL algorithm the bound can be raised by one.

The cores discovered between the points at which the bounds are raised will be referred to as an iteration. An example of how these iterations are defined can be observed in Figure 4.1, the solver finds κ^0 and raises the cost, thus creating Iteration 0. Followingly, it finds several cores, after determining κ^i the cost is raised once more and Iteration 1 thus consists of all cores found between the cores κ^1 and κ^i (including both κ^1 and κ^i in the Iteration). Finally, the cost is raised once more after finding the core κ^j and thus Iteration 2 consists of all cores found between κ^{i+1} and κ^j.

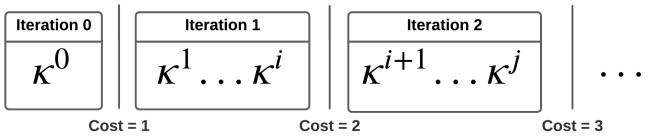


Figure 4.1: Visualization of IHS iterations

Example 7
Let's say that we have found the core $\{x_1, x_2\}$ using an IHS-solver, the bound is raised by one and we can create the cardinality constraint $x_1 + x_2 \leq 1$. Now the IHS-solver finds the core $\{x_2, x_3\}$ after which the bound is **not** raised and thus no translation takes place. In the next iteration of the IHS-solver, the core $\{x_1, x_3\}$ is found which raises the bound by one and thus a translation takes place. In this case, the OLL-core $\{x_3, x_1 + x_2 \leq 1\}$ resulting in the cardinality constraints $\{x_1 + x_2 \leq 2, x_3 + (x_1 + x_2 > 1) \leq 1\}$ is created.

Example 7 shows a potential translation strategy for simulating the OLL algorithm based on cores found by an IHS-solver (it should be noted that this is not the only approach with regard to translation). In general, it can be seen that the OLL-core allows all minimum hitting sets as possible solutions to the cardinality constraints. In a sense, if an OLL-solver were to be provided with the translated OLL-cores in such a manner, it would reject all minimum hitting sets as solutions in the subsequent iteration. However, as will be shown in Example 8, it is not viable to use such an approach in a practical setting.

Example 8
The following example shows that the approach of allowing all minimum hitting sets as solutions to the cardinality constraints is not applicable. In this example, Cores is the set of cores discovered by the IHS-solver since the last time the bound was raised, MHSes is a set of all the minimum hitting sets of the cores found up and until this iteration. Created Cardinality Constraints corresponds with the constraints produced by an OLL-core which allows all minimum hitting sets as solutions (e.g. $x_1 + x_2 + x_3 + x_4 \leq 1$ allows all minimum hitting sets $\{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}\}$ as solutions)

- **Iteration 0**
  Cores: $\{\{x_1, x_2, x_3, x_4\}\}$
  MHSes: $\{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}\}$
  Created Cardinality Constraints: $\{x_1 + x_2 + x_3 + x_4 \leq 1\}$
- **Iteration 1**
  Cores: $\{\{x_4, x_5\}, \{x_1, x_5\}\}$
  MHSes: $\{\{x_1, x_4\}, \{x_1, x_5\}, \{x_2, x_5\}, \{x_3, x_5\}, \{x_4, x_5\}\}$
  Created Cardinality Constraints: $\{x_1+x_2+x_3+x_4 \leq 2, x_5+(x_1+x_2+x_3+x_4 > 1) \leq 1\}$
- **Iteration 2**
  Cores: $\{\{x_1, x_2, x_6\}, \{x_2, x_5\}, \{x_3, x_5\}, \{x_4, x_6\}\}$
  MHSes: $\{\{x_2, x_4, x_5\}, \{x_2, x_5, x_6\}, \{x_3, x_5, x_6\}, \{x_1, x_4, x_5\}, \{x_4, x_5, x_6\}, \{x_1, x_5, x_6\}\}$
  Created Cardinality Constraints: $\{x_1 + x_2 + x_3 + x_4 \leq 2, x_5 + (x_1 + x_2 + x_3 + x_4 > 1) \leq 2, x_6 + (x_5 + (x_1 + x_2 + x_3 + x_4 > 1) > 1) \leq 1\}$
- **Iteration 3**
  Cores: $\{\{x_3, x_6\}, \{x_1, x_7\}, \{x_2, x_7\}\}$
  MHSes: $\{\{x_1, x_2, x_3, x_4\}, \{x_3, x_5, x_6, x_7\}, \{x_1, x_2, x_5, x_6\}, \{x_2, x_5, x_6, x_7\}, \{x_1, x_5, x_6, x_7\}, \{x_4, x_5, x_6, x_7\}\}$

A problem occurs, namely, in **Iteration 2** a MHS consisting of $\binom{\{x_1,x_2,x_3,x_4\}}{3}$ is not present and thus the created cardinality constraints do not allow this as a solution. However, in **Iteration 3**, the minimum hitting set $\{x_1, x_2, x_3, x_4\}$ is present but it is not possible to create an OLL-core based on previously created cardinality constraints such that it allows the minimum hitting set as a solution.

Example 8 demonstrates the inherent disconnect between how both approaches explore the solution space. An OLL-solver has the inherent restriction that any solution to the cardinality constraints is required to build upon solutions of the previous iterations by adding a single variable. IHS-solvers put no such restrictions on its hitting sets and, as can be seen in Example 8, its hitting sets are not (necessarily) influenced by hitting sets of previous iterations.

It should be noted that this does not imply that it is impossible to find cores which would allow an OLL-solver to create the appropriate cardinality constraints (e.g. include $x_1 + x_2 + x_3 + x_4 \leq y$

in each iteration in Example 8). Rather, the example shows the difficulty of determining whether a subset of variables is required to be included in an OLL-core without the usage of a SAT-solver.

4.2. Setting change to allow simulation

Section 4.1 exhibits the difficulty of the aforementioned simulation in a practical setting. In this section, the context in which such a translation is possible and the inner workings of an algorithm for performing the simulation are described.

As discussed in the previous section, there is an inherent disconnect between the potential MaxSAT solutions produced by an IHS-solver and an OLL-solver. The following assumption alleviates this issue and restricts the minimum hitting sets of the cores found by an IHS-solver to act similarly to the solutions of the cardinality constraints created by an OLL-solver.

Assumption 1
For every minimum hitting set hs^i of size y of the cores found up and until iteration i, there exists a minimum hitting set hs^{i-1} of size $y-1$ of the cores found up and until the previous iteration such that $hs^{i-1} \subset hs^i$. *It should be noted that this assumption requires that the final iteration contains all of the possible MCSes as MHSes.*

Assumption 1 enforces the structure of the minimum hitting sets to be identical to how OLL-solvers create solutions to cardinality constraints, by extending an existing solution by a single variable. In this manner, it becomes possible to freely translate between the two algorithms due to the identical fashion in which potential solutions are generated. The following section formalises the properties required of an algorithm for such a translation and specifies its inner workings.

4.2.1. Global Procedure & Properties

As specified in Section 2.2.1.3, the goal of an IHS-solver is to find a set of cores which refutes all of the possible minimum hitting sets of the core-trace of size y. The bound is raised when it achieves this aim, signifying that all minimum hitting sets of size y have been refuted and a hitting set of at least size $y+1$ is necessary to form an MCS. If there exists no set of cores refuting all minimum hitting sets of size y then this indicates that there is an optimal solution of size y, precisely the minimum hitting set which cannot be rejected.

To this end the proposed procedure takes the following three inputs:

1. A set of cardinality constraints $\mathcal{C}$ which have been created in previous iterations
2. The set of cores $\mathcal{A}$ found by an IHS-solver in previous iterations.
3. A set of cores $\mathcal{B}$ to translate which have been generated by an IHS-solver.

The procedure would then create two outputs:

1. An OLL-core κ_{OLL}
2. A resulting set of cardinality constraints $\mathcal{C}^*$ **after** incorporating κ_{OLL}

The main challenge of creating such a procedure is finding a core such that if an OLL-solver were to be run using the translated cardinality constraints then it would be able to find the optimal solution. To this end, the following property is proposed as being required of the OLL-cores (and their corresponding cardinality constraints) to guarantee the aforementioned optimality constraint.

Property 1
Let $\mathcal{C}$ be a set of cardinality constraints and *cores* be a set of cores of the original formula then $\forall hs \in \text{ALL-MinHS}(cores),\ \exists I \in \text{SOLUTIONS}(\mathcal{C})\ :\ \{x_i \mid x_i \in hs,\ I(x_i) \neq 1\} = \emptyset$.

Definition 9
All-MinHS($\mathcal{K}$) returns all of the possible minimum hitting sets of the set $\mathcal{K}$

Definition 10
SOLUTIONS($\mathcal{C}$) returns all possible solutions of the cardinality constraints present in the set $\mathcal{C}$.

Property 1 states that the solution space of the created cardinality constraint(s) contains all possible minimum hitting sets of a set of cores. ALL-MinHS and SOLUTIONS are defined in Definition 9 and Definition 10, respectively.

In combination with Assumption 1, an algorithm producing OLL-cores which possess Property 1 will allow an OLL-solver to find an optimal solution. To this end, it is first shown that adding a new core to an existing set of cores with a minimum hitting set of size y does not introduce any additional minimum hitting sets of the same size:

Lemma 1
Let $\mathcal{K}$ be a set of cores of the original formula with its corresponding MHS mhs, κ a core of the original formula not in $\mathcal{K}$ and $\mathcal{K}^* = \mathcal{K} \cup \{\kappa\}$ with corresponding minimum hitting set mhs^*. If $|mhs| = |mhs^*|$ then $mhs^* \in$ ALL-MinHS($\mathcal{K}$).

Proof. We know that for mhs and mhs^* to be a minimum hitting set, it should hold that $\forall \kappa' \in \mathcal{K} \ : \ mhs \ \cap \ \kappa' \neq \emptyset \wedge mhs^* \ \cap \ \kappa' \neq \emptyset$. However, by definition of mhs^*, we also know that $\kappa \ \cap \ mhs^* \neq \emptyset$.

Thus mhs^* is required to be an MHS of $\mathcal{K}$, if $mhs^* \notin$ All-MinHS($\mathcal{K}$) but $mhs^* \in$ All-MinHS($\mathcal{K} \ \cup \ \kappa$) then this would mean that mhs^* is not an MHS of $\mathcal{K} \cup \kappa$ as it does not hit all cores in $\mathcal{K}$ (otherwise it would be an element of All-MinHS($\mathcal{K}$)). Thus we arrive at a contradiction proving that $mhs^* \in$ All-MinHS($\mathcal{K}$). □

Lemma 1 indicates that the translation can be performed as soon as the core which raises the bound has been found. If Lemma 1 were to not hold then this would mean that the strategy would need to account for these additional hitting sets without prior knowledge of future cores. It will now be proven that if an algorithm produces OLL-cores with Property 1 then an OLL-solver will be able to find the optimal solution using these cores (under Assumption 1).

Theorem 1
If an algorithm produces OLL-cores based on a set of cores provided by an IHS-solver which possess Property 1, then an OLL-solver provided with these OLL-cores will be able to find the optimal solution(s) to the MaxSAT problem (working under Assumption 1).

Proof. We will prove this by contradiction. Let us assume that an algorithm has created a set of OLL-cores $\mathcal{K}_{OLL}$ (and corresponding cardinality constraints $\mathcal{C}$) with Property 1 based on the set of cores $\mathcal{A}$ found by an IHS-solver and that these OLL-cores do not allow an OLL-solver to find the optimal solution. We split into two cases:

1. In the first case, we assume that the set of cores of the IHS-solver provided to the OLL-solver has a minimum hitting set corresponding to an optimal solution. We know that the cardinality constraints based on the translated OLL-cores possess Property 1 meaning that one of the solutions to these cardinality constraints corresponds with the optimal solution. However, this leads to a contradiction to the assumption that the optimal solution could not be found with the set of created OLL-cores.
2. In the second case, we assume that the core-trace of the IHS-solver is "incomplete", meaning that $0 \leq |MinHS(\mathcal{A})| < opt$. In this case, the OLL-solver would be provided with $\mathcal{K}_{OLL}$ and continue its execution (it should be noted that this warm-start of the OLL-solver does not influence the inner workings of the OLL-solver thus retaining the properties related to the correctness of the algorithm). If the optimal solution can not be found then this would mean that the building blocks for a solution are not present in one of the translated OLL-cores of iteration i. However, we know that all solutions contain at least one of the variables present in the OLL-core by Assumption 1. This would lead to a contradiction as this would imply that there exists an optimal solution which contains fewer or different variables than the variables present in the MHSes of that iteration (which is untrue by Assumption 1 and the fact that the translated OLL-cores possess Property 1).

□

4.2.2. Algorithm Outline

The aim of the procedure is thus to attempt to find an OLL-core possessing Property 1. The most readily apparent manner in which to achieve this is by overestimating the solution space of the resulting cardinality constraints. This is based on the fact that all minimum hitting sets of the following iteration either contain a variable which has not been introduced in any of the previous cardinality constraints or it allows one more of the variables in existing cardinality constraints to be set to true (by Assumption 1).

All variables in a set of cores $\mathcal{K}$ are given by vars($\mathcal{K}$) and the variables in a cardinality constraint are produced using the same notation. The procedure then functions as follows:

Algorithm 4 Naïve Core Translation

Input: $\mathcal{C}$ **- Cardinality constraints of previous iterations**
$\mathcal{A}$ **- Cores translated in previous iterations**
$\mathcal{B}$ **- New cores to translate generated by an IHS-solver**
Output: κ_{OLL} **- OLL-core**
$\mathcal{C}'$ **- The resulting set of cardinality constraints**

1: **procedure** CreateOLLCore
2: $\kappa_{OLL} \leftarrow \{x_i \mid x_i \in \text{vars}(\mathcal{B}),\ x_i \notin \text{vars}(\mathcal{A})\}$
3: Cards $\leftarrow \{\text{Card}_i \mid \text{Card}_i \in \mathcal{C},\ \text{vars}(\text{Card}_i) \cap \text{vars}(\mathcal{B}) \neq \emptyset\}$
4: $\kappa_{OLL} \leftarrow \kappa_{OLL} \cup$ Cards
5:
6: $\mathcal{C}' \leftarrow \mathcal{C} \cup$ UpdateRHSAndCreateSum(κ_{OLL})
7: **return** κ_{OLL}, $\mathcal{C}'$

Algorithm 5 Sum Creation

1: **procedure** UpdateRHSAndCreateSum($\mathcal{C}$)
2: new$\mathcal{C} \leftarrow (\sum_{x_i \in \text{CardsToLits}(\mathcal{C})} \neg l_i \leq 1)$
3: **for** $c \in Cards(\mathcal{C})$ **do**
4: $rhs(c) \leftarrow rhs(c) + 1$
5: new$\mathcal{C} \leftarrow$ new$\mathcal{C} \cup \{c\}$
6: **return** new$\mathcal{C}$

Algorithm 4 starts by finding all of the variables present in $\mathcal{B}$ which have not been found in a core thus far and adding them to κ_{OLL} (Line 2). It then finds all of the cardinality constraints which contain any of the variables in $\mathcal{B}$ and adds them to the core in Line 3. Finally, it creates the cardinality constraint corresponding to this core in Line 6.

The function UpdateRHSAndCreateSum, shown in Algorithm 5, creates a new sum specifying that one additional variable in $\mathcal{C}$ can be set to true and the right-hand side of all the original cardinality constraints is increased by one. It is assumed that the function CardToLits($\mathcal{C}$) returns the literals representing the cardinality constraints of $\mathcal{C}$. Furthermore, it is assumed that $Cards(\mathcal{V})$ returns the cardinality constraints present in the set $\mathcal{V}$.

Theorem 2
The cores and cardinality constraints produced by Algorithm 4 possess Property 1

Proof. This is proven by induction.
Base Case:
The base case is when $\mathcal{A} = \mathcal{C} = \emptyset$ and $\mathcal{B}$ is any core. In this case, the procedure would add all literals to κ_{OLL} and the resulting cardinality constraint would necessarily allow all hitting sets of size one thus the property holds.

Induction Hypothesis:
We assume that for some iteration i, the solution space of the created cardinality constraints $\mathcal{C}^i$ of κ^i_{OLL} allows all minimum hitting sets of $\mathcal{A}$. We want to prove that the same holds for κ^{i+1}_{OLL} and $\mathcal{C}^{i+1}$.

So we know that $\forall hs \in \text{ALL-MinHS}(\mathcal{A}),\ \exists I \in \text{SOLUTIONS}(\mathcal{C}^i)$ s.t. $\{x_i \mid x_i \in hs,\ I(x_i) \neq 1\} = \emptyset$. Now the procedure is provided with a new set of cores $\mathcal{B}$ and it creates a new core κ^{i+1}_{OLL} and we need to show that $\mathcal{C}^{i+1}$ allows all of the minimum hitting sets of $\mathcal{A} \cup \mathcal{B}$ as solutions. By Assumption 1, we know that every solution is extended by a single variable which either is already part of a previous cardinality constraint in $\mathcal{C}^i$ or it is a new variable.

1. In the case of a new variable it is added to the core and allowed to be added to the existing set of solutions to $\mathcal{C}^i$, thus $\mathcal{C}^{i+1}$ allows all minimum hitting sets which have solely added a newly introduced variable to an existing solution.
2. In the case of a variable being added to a solution which was already part of a cardinality constraint present in $\mathcal{C}^i$, the procedure adds all of the cardinality constraints which contain this variable to κ^{i+1}_{OLL}. This means that the resulting set of cardinality constraints $\mathcal{C}^{i+1}$, necessarily allows this variable to be added to any of the existing solutions (if this were not to be the case then this would be due to a constraint containing the variable which is not part of the core but this constraint would by construction be added to κ^{i+1}_{OLL}).

For all variables in vars($\mathcal{B}$), either case 1 or case 2 is applicable. This shows that the cores and cardinality constraints produced by Algorithm 4 possess Property 1. □

Thus it has been proven that the OLL-cores produced by Algorithm 4 possess Property 1. The following example shows the workings Algorithm 4.

Example 9
Let's say that as an input we have received the following: $\mathcal{C} = \{x_1 + x_2 \leq 1\}$ and $\mathcal{A} = \{\{x_1, x_2\}\}$. We want to translate the core set $\mathcal{B} = \{\{x_3, x_1\}, \{x_3, x_2\}\}$ and want to find an OLL-core which covers these cores. We first determine $\{x_i \mid x_i \in \text{vars}(\mathcal{B}),\ x_i \notin \text{vars}(\mathcal{A})\} = \{x_3\}$. Now according to the procedure, we want to find the set of cardinality constraints intersecting with the variables $\{x_1, x_2\}$. So we look at all constraints (in this case only the cardinality constraint $\mathcal{C}_1 = x_1 + x_2 \leq 1$) and see that only this constraint $\mathcal{C}_1$ should be included in the core. Thus we have found the OLL-core $\{x_3,\ x_1 + x_2 \leq 1\}$, creating the set of cardinality constraints $\{\{x_3 + \neg(x_1 + x_2 \leq 1) \leq 1,\ x_1 + x_2 \leq 2\}\}$.

4.2.3. Core Minimization

The algorithm presented in Section 4.2.2 creates a core possessing Property 1, however, this core does not represent a MUS. The procedure might overestimate the cardinality constraints/variables required to be present for the core(s) to possess Property 1. To this end, a procedure analogous to core minimization is introduced (an overview of such strategies can be seen in [30]). These techniques aim at determining a minimum set of clauses such that they retain the properties related to being a core. The procedures oftentimes involve multiple calls to a SAT-solver to check for (un)satisfiability of proposed cores of smaller cardinality. These algorithms can be adapted to create a pseudo-MUS of the cores created by Algorithm 4 without the usage of SAT-calls. In this case, these procedures would rely on a check of whether all minimum hitting sets are part of the solution space of the resulting cardinality constraints of the OLL-core.

4.3. Chapter Conclusion

In conclusion, it is difficult to directly simulate OLL based on IHS due to the intrinsic differences between the manner in which the solution space is explored by both approaches. The arduous nature of determining which variables are required to be in a solution in such a manner that an OLL-solver can utilise this information further exhibits this difficulty. However, as shown in this chapter, if the setting under which the simulation is changed to assuage this disconnect, then it is possible to perform the simulation in a straightforward manner.

In general, this means that enforcing a ”similar” execution trace between OLL and IHS based on the execution trace of an IHS-solver is a strenuous task. This limits the efficacy of an approach which relies on information sharing between the two algorithms. However, the awareness of this impediment allows the creation of an algorithm which circumvents these obstacles.

5

Hybrid Algorithm

This section will discuss the techniques underlying the proposed hybrid OLL-IHS MaxSAT algorithm. It will also discuss practical considerations related to implementing such an algorithm.

5.1. General Intuition

One iteration of an OLL-solver can correspond to many iterations of an IHS-solver but the reverse is not necessarily true. As shown in Section 2, the lower bound on the solution cost of an OLL-solver is raised in each iteration of the algorithm while this is not inexorably the case for an IHS-solver. Thus there is not guaranteed to be a mapping from a single iteration of an IHS-solver to a single iteration of an OLL-solver, however, there might exist a mapping of multiple iterations of an IHS-solver to a single iteration of an OLL-solver while avoiding the aforementioned issues with regards to simulation (as described in Section 4.1).

As mentioned in Section 4, it is difficult to perform a direct simulation between the OLL and IHS algorithms. Nevertheless, it is possible to translate the information found by an IHS-solver to an OLL-solver to improve its efficiency. The intuition behind the potential increase in efficacy relies on the fact that the cores of the original formula generated by an IHS-solver oftentimes result in a different structure than the OLL-cores found by an OLL-solver. A difference in solver behaviour occurs when this information is provided to an OLL-solver which potentially avoids the intricate nature of complex cardinality constraints thus increasing the performance in subsequent SAT-calls. This decrease in complexity oftentimes propagates to subsequent iterations. Determining whether this information sharing is beneficial is related to the open question of whether an OLL-solver can benefit from being provided with an initial set of cores.

5.2. Partially Completed Issue

An issue with this translation would be how to determine when to transfer a ”completed” constraint to the OLL-solver. For example, let’s say that the optimal cost of the current formula is four and the accompanying constraints would be $\mathcal{C} = \{x_6 + x_7 + x_8 \leq 2, x_1 + x_2 + x_3 \leq 2\}$. It can be seen in Example 10 that the bound is raised when the core $\{x_1, x_3\}$ is added to the list of cores (as it is disjoint from the previous cores). However, as can be observed from the set $\mathcal{C}$, the second cardinality constraint is not yet completed and would require two more cores to complete the cardinality constraint (namely the cores $\{\{x_2, x_3\}, \{x_1, x_2\}\}$). It is thus difficult to determine what constraint is said to be ”completed” when the bound is raised by the IHS-solver.

Example 10
Consider that an IHS-solver has generated the following set of cores:
$\{\{x_7, x_8\}, \{x_6, x_8\}, \{x_6, x_7\}, \{x_1, x_3\}\}$ a minimal hitting set would be $\{x_8, x_6, x_1\}$. The solution cost is thus equal to three, these constraints can be converted to two cardinality constraints: $\{x_6 + x_7 + x_8 \leq 2, x_1 + x_3 \leq 1\}$ mapping four iterations of IHS to two iterations of OLL.

However, this completion issue is not present if the cores found by the IHS-solver are used as a starting point for the OLL-solver (i.e. running the IHS-solver for a certain number of iterations and then transferring the acquired knowledge to the OLL-solver and continuing execution from that point onwards). This starting point might contain partially completed cardinality constraints but these would be used to create new constraints (though perhaps more complex than the aforementioned set of two cardinality constraints) when executing the OLL algorithm. This issue serves to demonstrate a potential impediment when performing a translation, namely, that these partially completed constraints could lead to a larger number of cardinality constraints than required if purely the OLL-solver was utilised to determine the constraints from the beginning (as can be seen in Example 11). This illustrates that how many and which cores are generated is of the utmost importance to the performance of a hybrid OLL-IHS algorithm.

Example 11
Let's say that we have found the cores $\{\{x_7, x_8\}, \{x_6, x_8\}, \{x_6, x_7\}, \{x_1, x_3\}\}$ using an IHS-solver and we have created the cardinality constraints $\{x_6 + x_7 + x_8 \leq 2,\ x_1 + x_3 \leq 1\}$. Now we run the solver and find the following core: $\{x_2, x_1 + x_3 \leq 1\}$, this would lead to the following set of constraints: $\{x_6 + x_7 + x_8 \leq 2, x_1 + x_3 \leq 2,\ x_2 + (x_1 + x_3 > 1) \leq 1\}$ leading to more cardinality constraints than the set $\{x_6 + x_7 + x_8 \leq 2,\ x_1 + x_2 + x_3 \leq 2\}$.

In the following sections, two techniques for creating a translation from cores of the original formula to OLL-cores are proposed, one based on exploiting substructures in the hypergraph (see Definition 11) of the cores and the other based on extracting disjoint cores.

Definition 11
A **hypergraph** is a generalization of a graph which allows a single edge to connect multiple vertices (a so-called hyper-edge). In the context of MaxSAT, the vertices represent soft clauses and the edges represent cores. It is possible to convert from a hypergraph to a regular graph by translating each hyper-edge to a regular edge by creating a complete graph between each vertex within a hyper-edge.

5.3. Pipeline Structure

The main structure of the algorithm follows several phases which together constitute a singular MaxSAT algorithm. An overview of this pipeline can be seen in Figure 5.1.

1. **Core Generation Phase:** This phase consists of generating cores of the original formula. This can be done using specific MUS enumeration algorithms (such as [33] and [34]). However, initial preliminary testing shows that, given certain budgets, the cores generated by these algorithms are less diverse than the ones generated by utilizing an IHS-solver. Due to these practical considerations, in this work, an IHS-solver is employed to generate the cores of the original formula. Nevertheless, it should be noted that any algorithm which generates cores of the original formula can be used.
2. **Core Translation Phase:** This phase translates the cores of the original formulas to OLL-cores such that it is able to find an optimal solution using these cores. Due to the issues described in Section 4.1, these cores are required to not disallow possible solutions. For this translation, the techniques described in Section 5.4 and Section 5.5 are utilised.
3. **Final Solving Phase:** In this phase, the OLL-cores found during the Core Translation Phase are provided to the OLL-solver which is then tasked with solving the instance to optimality. `RC2` is used as the OLL-solver as it is well-documented and simple to adapt.

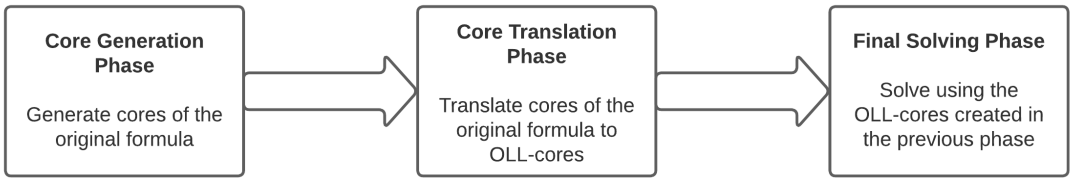


Figure 5.1: Hybrid Algorithm Pipeline

5.4. Mixture

The structure of the hypergraph of the set of cores in combination with the information provided by the MHS of this set of cores can be exploited to create a set of cardinality constraints for the OLL-solver. The foundation of this technique relies on the fact that adding additional cores to a set of cores does not reduce the size of the minimum hitting set of a connected component (see Definition 12) in the hypergraph (as it would otherwise have been possible in prior iterations to create a minimum hitting set of a smaller cardinality). It is important to note that due to the issues mentioned in Section 4.1, it is burdensome to directly generate complex constraints as it is impossible to predict which subsets of variables can be excluded from the solution space solely based on the cores found by the IHS-solver. Acknowledging this fact, the following techniques are proposed for extracting cardinality constraints from the hypergraph.

Definition 12
A **connected component** in a hypergraph is a set of hyper-edges which are connected to one another via a path. Thus, for every hyper-edge in the connected component, it is possible to reach each other hyper-edge in the connected component by means of overlapping hyper-edges.

5.4.1. Hypergraph Technique

The simplest manner in which to convert a connected component in the hypergraph to a cardinality constraint usable by an OLL-solver is to determine the size of the minimum hitting set of the component and utilise that as the right-hand side of the cardinality constraint. This encodes the information that no solution to the MaxSAT instance can contain fewer variables of the connected component than indicated by the minimum hitting set (as otherwise a minimum hitting set of smaller cardinality would be possible). More formally, given a set of variables of a connected component $\mathcal{V}$ and a minimum hitting set of the cores in this component with size y, this technique would create the constraint $\sum_{x_i \in \mathcal{V}} x_i \leq y$. The functioning of such an approach can be seen in the following example.

Example 12
Let's say that an IHS-solver has found the following set of cores: $\{\{x_1, x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}\}$ which has a minimum hitting set of size two. We could then create the constraint $x_1 + x_2 + x_3 + x_4 + x_5 \leq 2$ which encapsulates the cores found by the IHS-solver.

Example 12 shows a manner in which the connected components of the hypergraph can be used to determine a cardinality constraint usable by an OLL-solver. This constraint avoids the simulation issue by allowing all possible subsets of variables of size y as solutions. However, this exhibits the main drawback of this technique; it might allow more solutions than are necessary to prove optimality. This induces increased strain upon the SAT-solver as it is required to examine each solution as a possible MCS. This issue becomes increasingly apparent as the size of the connected component grows since the number of possible solutions to the resulting cardinality constraint increases exponentially.

5.4.2. Lower Bound Accuracy

When translating constraints there is a trade-off to be made concerning the lower bound found by the IHS-solver and the translated lower bound. If the Core Generation Phase is interrupted **after**

finding a new core but **before** having calculated a new MHS then there is a potential disconnect betwixt the true lower bound of the cores which are covered by the cardinality constraint and the right-hand side of said constraint. This issue can be alleviated by completing the unperformed call to the MHS solver, however, as the bound of the constraint is nevertheless valid, it is possible to disregard this call. This avoidance of an additional call to the MHS solver can lead to improved solving times depending on the performance of the SAT-oracle.

5.4.3. Clique Technique

Definition 13
A **clique** is a structure in a graph in which each node in the clique is connected to every other node in the clique by an edge.

One of the caveats of the constraints generated by the technique discussed in Section 5.4.1 is that it potentially requires an expensive call to an MHS algorithm (as discussed in Section 5.4.2). To alleviate this issue, certain structures allow the right-hand side of the cardinality constraint to be easily extracted. An example of this is when there exists a clique (see Definition 13) in the hypergraph consisting of cores of size two. In this case, the minimum hitting set problem devolves into the vertex cover problem which can be solved in polynomial time if the problem instance consists of a single clique. In this case, any subset of $\binom{vars}{|vars|-1}$ suffices as a solution and thus the right-hand side of the created constraint is equal to the number of variables in the connected component minus one. This strategy can be seen in the following example.

Example 13
Let's say an IHS-solver has found the following cores: $\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_1\}$ which correspond with a clique consisting of three variables (x_1, x_2 and x_3). Due to the fact that it is a clique consisting of cores of size two, we can determine that the cardinality constraint should be $x_1 + x_2 + x_3 \leq 2$

If this structure occurs then it is furthermore guaranteed that no excessive solutions exist as all combinations of variables are potential minimum hitting sets (i.e. potential minimum correction sets). Checking whether a set of vertices forms a clique can be performed in polynomial time, thus avoiding the superfluous time complexity of a maximum clique algorithm. Finding a maximum clique is thus straightforward due to the fact that the technique presented in Section 5.4.1 is solely concerned with whether a connected component in the hypergraph is a clique.

5.4.4. Hypergraph Cutting

The main issue of the technique mentioned in Section 5.4.1 is that the cardinality constraints which it generates can depend on a large number of variables which leads to copious amounts of additional clauses and variables. To this end, an adjustment to the aforementioned naïve technique is proposed which attempts to ameliorate this issue.

The intuition behind this improvement relies on the separability of different components of the hypergraph. If the hypergraph consists of two components which are sparsely connected (e.g. two clusters of variables with a single edge between the clusters as can be seen in Figure 5.2) then perhaps it would be preferable to create a more complex constraint to encapsulate this structure rather than creating one sizeable constraint which loses this structural information. The main aim is thus to find these clusterings in such a manner that the sparsely connected components are separated while densely clustered components remain intact. This problem can be seen as similar to the community detection problem [35] in which so-called "communities" are vertices which are tightly bound to one another but loosely bound to vertices of other communities. These communities in the hypergraph could indicate variables which are likely to be found together in (future) cores.

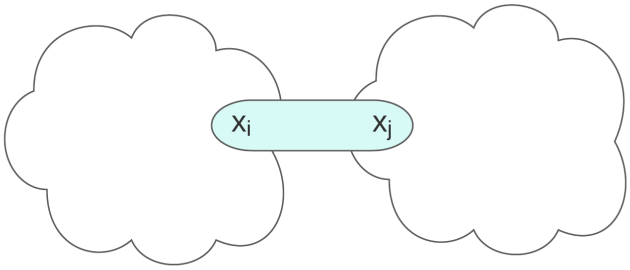


Figure 5.2: Example of two communities connected by a single hyper-edge

Thus, if the connected component is of a certain size then it would be desirable to diminish the magnitude of the overarching cardinality constraint by splitting the graph based on these communities. One of the issues involved with splitting the hypergraph is retaining the information concerning the lower bound. The procedure thus splits the graph into two (or more) connected components which facilitates the calculation of the minimum hitting sets of the components individually, disregarding the cores between components in said calculation. These minimum hitting sets provide a lower bound for each of the components (similar to Section 5.4.1).

If an initial lower bound of a connected component in the hypergraph was found to be y by the IHS-solver and after separating the component a sum of lower bounds of these communities is found to be s, then a difference of $y-s$ is required to be encoded. If $y-s=0$ then no additional steps are necessary as the lower bound has been properly encoded using the current connected components. However, if $y-s>0$ the lower bound of the connected components does not correspond with the lower bound which has been found during the Core Generation Phase. While encoding this difference is not required for the correctness of the algorithm, it serves to retain the lower bound found by the IHS-solver rather than discarding this information.

To this end, new constraints are created that encode these lower bounds while acknowledging that no assumptions can be made concerning which variables are required to be additionally allowed to be set to true. The goal is thus to create the OLL-cores in such a way that it allows $y-s$ additional variables to be set to true. An example of this strategy can be seen in Example 14. The pseudocode in Algorithm 6 illustrates this process.

Example 14
Let's say we have found a hypergraph consisting of the following hyper-edges: $\{\{x_1,x_2,x_3,x_4\}, \{x_5,x_6,x_7,x_8\},\{x_4,x_5\},\{x_3,x_6\},\{x_8,x_9\}\}$ which has an MHS of size 3. We then find a partition of nodes $CL_1=\{x_1,x_2,x_3,x_4\}, CL_2=\{x_5,x_6,x_7,x_8,x_9\}$ and calculate the edges which are *fully* contained within either of these partitions: $Edges(CL_1)=\{\{x_1,x_2,x_3,x_4\}\}$, $Edges(CL_2)=\{\{x_5,x_6,x_7,x_8\},\{x_8,x_9\}\}$. We can see that $|MinHS(Edges(CL_1))|=|MinHS(Edges(CL_2))|=1$.

Now we create the cardinality constraints $\mathcal{C}=\{x_1+x_2+x_3+x_4\leq 1, x_5+x_6+x_7+x_8+x_9\leq 1\}$. Since $y=3$ and $s=|MinHS(Edges(CL_1))|+|MinHS(Edges(CL_2))|=2$, it can be observed that $y-s=1$ thus indicating that the original lower bound is not fully encoded by $\mathcal{C}$. To alleviate this issue, we add another constraint allowing one more variable in either CL_1 or CL_2 to be set to true resulting in the following set of cardinality constraints: $\mathcal{C}'=\{x_1+x_2+x_3+x_4\leq 2, x_5+x_6+x_7+x_8+x_9\leq 2, (x_1+x_2+x_3+x_4>1)+(x_5+x_6+x_7+x_8+x_9>1)\leq 1\}$. This final set of cardinality constraints fully encodes the lower bound of three.

Algorithm 6 Hypergraph Cutting

Input: *Component* **- The connected component to cut**
Input: *mhs* **- Minimum hitting set of** *Component*
Output: $\mathcal{C}$ **- Set of cardinality constraints encoding lower bound of** *Component*

```
1:  procedure CutHypergraph
2:      C ← ∅
3:      new_lb ← 0
4:      lits ← ∅
5:      S ← CreateCommunities(Component)
6:      for community ∈ S do
7:          lb ← MinHS(community)
8:          if lb == 0 then
9:              lits ← lits ∪ community
10:         else
11:             new_lb ← lb + new_lb
12:             C ← C ∪ {∑_{x_i ∈ community} ≤ lb}
13:     for i ∈ [0, |mhs| − new_lb) do
14:         if i = 0 then
15:             C ← C ∪ lits
16:         C ← UpdateRHSAndCreateSum(C)
17:     return C
```

Line 5 in Algorithm 6 creates the communities of the hypergraph component (more details concerning this clustering are discussed in the following paragraphs). The algorithm then loops over the resulting communities and creates cardinality constraints (in the same manner as specified in Section 5.4.1) while keeping track of the newly found lower bound of all communities. It then accounts for the discrepancy between the original lower bound ($|hs|$) and the new lower bound (new_lb) by iteratively calling the function UpdateRHSAndCreateSum in Line 16.

It should be noted that it is possible to get a lower bound of zero on a component if there are no edges between vertices within a connected component. In this case, the variables within this component are not added when creating the initial constraint but they are added when the first complex constraint is created (Line 16). This ensures that these variables are taken into consideration without making any presumptions about the combination of variables required to be present in a solution.

The clustering in Line 5 can be performed using a multitude of algorithms (such as [36], [37]). In this work, Kumar's algorithm as discussed in [38] is utilised as this is the method implemented in HyperNetX[1].

5.4.5. Core Rejection

An additional problem which could occur during the translation process is that the cores generated by the IHS-solver are substantially large as to introduce excessively sized cardinality constraints. To this end, a technique called core rejection is introduced, a simple but effective strategy which prevents the translation of connected components under certain conditions. Whenever the size of a connected component grows too substantial, it is rejected from consideration for translation. The intuition behind this technique is based on the fact that the translation cannot (directly) make use of complex cores due to the translation issue. However, this is not the case in the Final Solving Phase since the OLL-solver is free to create the constraints in any way it deems appropriate (provided with the initial building blocks). This indicates that in certain cases, the OLL-solver can reduce the size of the cores found by creating complex constraints rather than making use of the sizeable translated constraints.

The next step is to determine in which scenarios cores should be rejected. For certain instances all cores are large and the benefit of rejecting cores is diminished due to the fact that the OLL-solver is likely to find the same rejected cores as have been found by the IHS-solver leading to the

[1] https://github.com/pnnl/HyperNetX

introduction of unnecessary overhead for this search process. The simplest approach is to reject cores of a constant size (e.g. consisting of y variables). However, this approach does not take into consideration the structure of the instances which can have a substantial influence on the performance of the algorithm. To this end, this work proposes rejecting based on the size of the connected component relative to the number of soft clauses (as these soft clauses provide an upper limit on the size of the connected component). Initial experimentation was conducted using core rejection which rejects cores if their size exceeds 5% of the number of soft clauses, the results of which are discussed in Section 6.3.

5.5. Subset Selection

The Subset Selection approach takes inspiration from the disjoint core technique (also known as weight-aware core extraction on weighted instances [29]) in which rather than reformulating the instant a new core is found, it is postponed in an attempt to allow early termination of the algorithm. In the context of unweighted instances, this technique corresponds with a so-called disjoint core technique (additionally, a disjoint core phase is present in [10]) in which a number of disjoint cores are extracted before reformulation. This technique shows that it is feasible to make use of disjoint cores of the original formula to generate initial constraints for an OLL-solver. To this end, a technique is proposed which makes use of this fact by selecting a subset of cores produced by an IHS-solver to determine a lower bound on the solution cost.

In general, the aim is to generate a subset of cores which are disjoint from one another such that $S \subseteq \mathcal{A}$ where $\mathcal{A}$ is a set of cores produced by an IHS-solver. An intuitive goal of this selection is to encode the lower bound found by the solver as closely as possible, i.e. $|\text{MinHS}(\mathcal{A})| - |S|$ is minimal. The following example shows a potential selection of such subsets.

Example 15
Let's say that an IHS-solver has found the following cores: $\mathcal{A} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_1, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}\}$ which has a solution cost of three. We could select the set $S = \{\{x_1, x_2\}, \{x_3, x_4\}\}$ which would encode a lower bound of two using disjoint cores.

As can be seen in Example 15, the selected subset does not properly encode the lower bound as found by the IHS-solver. However, the IHS-solver might find cores which cannot be found using a single disjoint core phase and would require a multitude of disjoint core phases (of increasing complexity if existing cores were to be excluded from being discovered). There are several factors which influence the efficacy of the generated disjoint cores as different cores might lead the OLL-solver to perform better/worse due to the quality of the building blocks.

The main factor to consider is the manner in which the subset selection takes place. Intuitively, a higher lower bound is preferable due to the prevention of superfluous iterations to reject solutions of a certain cost which could have been prevented by utilizing a different selection of disjoint cores. A number of selection strategies are proposed and analyzed.

1. **Minimum Overlap** - This selection strategy (partially) considers the effect of choosing a core on future selection by making use of a greedy heuristic which focuses on locally optimal solutions. The aim is to minimize the number of cores removed from consideration by the choice of a core. This could potentially lead to more disjoint cores being selected which would improve the generated lower bound.
2. **Time-based** - This selection strategy uses a greedy approach to minimize the potential overhead of a selection procedure. The strategy works by iteratively adding cores during execution only if it does not overlap with any of the cores found thus far. To a certain extent, this strategy is reliant on the order in which the cores are generated and it can thus perform similarly to a randomized selection strategy.
3. **Smallest Size** - This selection strategy utilizes the fact that, in general, smaller core structures are preferred compared to large ones. This is due to the fact that smaller cores oftentimes lead to less overhead in terms of the number of variables/clauses introduced by the cardinality constraints. For IHS-solvers, the size of the clauses influences the performance of the minimum hitting set subroutine, as proving optimality potentially requires exploring

more branches if the generated cores are large. However, it should be noted that this strategy does not take into account how its selection affects the cores selected in further iterations.

5.6. Practical Considerations

5.6.1. MHS Calculation

An integral component of the IHS-solver used during the Core Generation Phase is the manner in which the MHS is calculated. [10] translates the MHS problem into a MIP formulation which can be utilised by a MIP solver (CPLEX in the case of `MaxHS`) to efficiently determine the MHS of the provided instance. However, as described in [10, Section 4.1.1], it is possible to use a specialized Branch and Bound algorithm to determine these MHSes. The usage of such a specialized algorithm allows increased controllability over the behaviour of the solver, permitting additional optimizations based on insights related to MaxSAT solving.

In this work, the Gurobi solver [39] is used to calculate MHSes.

5.6.2. IHS with non-optimal hitting sets

[10, Chapter 5] describes the usage of non-optimal hitting sets to ameliorate the copious time spent within the minimum hitting set solver. This approach can be incorporated within the aforementioned mixture algorithm as the cores generated by the SAT-oracle used in this method are nevertheless cores of the original formula. Intuitively, this version of IHS has the potential to generate an increased number of cores due to the non-optimality of the hitting set subroutine in addition to an overall improvement in the quality of the cores. This approach has been shown to expend more time in calls to the SAT-oracle than the ”regular” IHS algorithm and this thus exhibits a trade-off between the approaches. The practicality of the non-optimal version of IHS in comparison to the ”regular” approach is discussed in Section 6.4.

When implementing the IHS algorithm with non-optimal hitting sets, the main consideration is the manner in which these hitting sets are calculated. [10, Section 5.2.1] describes several techniques for calculating such hitting sets. Due to the non-optimality, these techniques oftentimes involve heuristics to determine the variable with which to extend the current hitting set.

In this work, a frequency-based heuristic is utilised which selects the variable which has occurred the most often in a core. This strategy has shown improved performance compared to a randomized selection strategy.

5.6.3. Realizable Minimum Hitting Sets

A further consideration is the enforcement of realizability for MHSes as discussed in [10, Section 4.1.1]. [10] utilises a special-purpose Branch and Bound algorithm to ensure that MHSes which are not realizable are disallowed.

The MIP-solver which has been used in this work (as described in Section 5.6.1), unlike CPLEX, does not allow the rejection of potential solutions using callbacks. Due to this fact, the realizability of MHSes is not considered in this work.

5.6.4. Memory Usage

Another point of consideration is the memory usage of the algorithm. Depending on which technique is used the amount of memory consumption can vary widely. Generally, the hybrid algorithm is likely to consume more memory than `RC2` due to the overhead introduced by the MIP model in combination with the additional memory required for techniques such as the community detection utilised by the hypergraph cutting described in Section 5.4.4. This increased memory usage is prone to lead to out-of-memory errors which negatively impact the performance of the hybrid algorithm in terms of the number of instances solved.

5.7. Chapter Conclusion

In conclusion, this work proposes a hybrid algorithm consisting of initially determining cores of the original formula by utilising an IHS-solver. After a predetermined duration of generating the cores of the original formula, these cores are translated to OLL-cores which can be utilised by an

OLL-solver. These translated OLL-cores are then provided to an OLL-solver for the final phase of the algorithm in which the optimal solution to the MaxSAT instance is determined.

Two translation techniques are proposed, namely the Subset Selection technique in which disjoint cores are selected for translation and the Mixture technique which utilises the lower bound on the connected component determined by the minimum hitting set to create a cardinality constraint. Furthermore, the Mixture technique is extended by two adjustments, the first being hypergraph splitting in which a single connected component is split into two or more subcomponents to prevent prodigiously sized cardinality constraints. The second such adjustment is core rejection which aims to address the same issue as the previous adjustment in an alternative manner, relying on the fact that an OLL-solver can create complex cardinality constraints which curtail the overhead introduced by unnecessarily sizeable OLL-cores.

Finally, certain practical considerations are discussed which could influence the efficacy of the proposed hybrid algorithm. These considerations relate to the manner in which the minimum hitting sets are calculated, whether IHS with non-optimal hitting sets is considered, the realizability of the minimum hitting sets and the amount of memory used by the hybrid algorithm.

6

Experimentation

This section will detail the results of the experimentation, comparing the different techniques specified in Section 5 with the OLL-solver `RC2`. Moreover, it will discuss an analysis of the instances based on a variety of metrics and propose a portfolio strategy for determining which technique to use to solve an instance.

6.1. Experimentation Setup

The experimentation was performed on the TU Delft supercomputer DelftBlue [40]. The experiments were performed in Red Hat Enterprise Linux 8 using an Intel XEON E5-6248R 24C 3.0GHz processor with 192 GB of memory. Each instance made use of a single core with 4GB of memory and a timeout of 3600 seconds. The unweighted benchmarks from the Maxsat Evaluation 2022[1] were used to analyse the performance of the hybrid algorithm.

The solvers used during experimentation were `RC2` (adjusted so that it does not convert the instance to a weighted instance), Subset Selection (Section 5.5) and the Mixture algorithm (Section 5.4) in combination with the following techniques:

1. **Community** - This includes the hypergraph cutting as described in Section 5.4.4 and the Core Rejection as described in Section 5.4.5. The default threshold for when to split a component or reject a core is when its size exceeds 5% of the number of soft clauses. Core Rejection is not applied to the cores created by the hypergraph splitting.
2. **CR** - Core Rejection as described in Section 5.4.5, the default threshold for rejecting a core is 5% of the number of soft clauses.
3. X**CT** - Indicates that the threshold for when hypergraph splitting occurs is X%. For example, the method Community$_{1CT}$ will perform hypergraph splitting if the size of a component exceeds 1% of the number of soft clauses.
4. X**s** - Indicates the amount of time spent in the Core Generation Phase. For example, Mixture$_{720s}$ will spend 720 seconds in the Core Generation Phase. The default amount of time spent in the Core Generation Phase is 180 seconds.
5. **NI** - Indicates that the SAT-solver used does not use incrementality when solving.
6. **0MHS** - Indicates that the Mixture algorithm allows connected components with a lower bound of 0 to be created.
7. **NO** - Indicates that the Core Generation Phase makes use of IHS with non-optimal hitting sets as opposed to ”regular” IHS.

For the Subset Selection technique of Section 5.5, the following notation is used:

- Minimum Overlap - Subset$_{MO}$
- Smallest Size - Subset$_{SmS}$

[1] https://maxsat-evaluations.github.io/2022/

- Time-based - Subset$_{TB}$

The hybrid algorithm in combination with the Subset Selection techniques makes use of a non-incremental SAT-solver by default.

Finally, the performance of the so-called *virtual best solver (VBS)* is presented. For each instance, the VBS represents the technique with the shortest solving time.

6.2. Results Summary

Generally, the techniques introduced in Section 5 show promise in terms of improving performance, specifically on certain instances. All of the techniques previously discussed perform competitively in comparison to `RC2`.

The Mixture technique (i.e. the hypergraph technique **without** hypergraph splitting) appears to perform well in combination with core rejection and a non-incremental SAT-solver. However, the best performance is gained by running the Core Generation Phase for 720 or 1800 seconds instead of the default 180 seconds. This indicates that gathering cores for a longer period of time can be beneficial up to a certain point after which the effects of the trade-off between the information gained by the Core Generation Phase and the amount of time spent within the Final Solving Phase become readily apparent. A detailed analysis can be found in Section 6.3.1.

The Community technique (i.e. the hypergraph technique **with** hypergraph splitting and core rejection) seems to perform marginally worse compared to its aforementioned counterpart. The best-performing technique consists of a combination of a non-incremental SAT-solver and a moderately increased threshold for when to perform the hypergraph splitting (compared to the default value of 5%). In this case, running the Core Generation Phase for a prolonged period of time results in substandard performance due to numerous out-of-memory errors encountered during the hypergraph splitting (induced by the additional representation of the potentially substantial hypergraph required for clustering) leading to certain instances not being solved. To further quantify this behaviour, Community$_{NI+10CT}$ encounters a total of 14 OOM errors, while the Community technique with a Core Generation Phase of 720 seconds encounters 19 OOM errors and a Core Generation Phase of 1800 seconds encounters 28 OOM errors. A detailed analysis can be found in Section 6.3.2.

The Subset Selection technique performs consistently well with two out of the three selection strategies solving marginally more instances than `RC2`. The strategies which take into account how the current selection of a core influences subsequent selections perform marginally better than the selection strategy which does not take such considerations into account. Interestingly, the VBS for this technique does not solve considerably more instances than the individual strategies indicating that there is a significant amount of overlap between the instances solved by the different strategies. A detailed analysis can be seen Section 6.3.3.

In an overall comparison, the Mixture technique and Subset Selection technique both slightly outperform `RC2`. Core Rejection and non-incremental SAT solving are both beneficial for the performance of the hybrid algorithm. For the Mixture technique, an extended Core Generation Phase is favourable for the performance while it is detrimental to the performance of the Community Technique (due to memory constraints). Finally, for the Subset Selection technique, a selection strategy which makes use of heuristics to determine which core to select performs the best.

Moreover, using IHS with non-optimal hitting sets instead of ”regular” IHS improves the performance of certain techniques. Especially the Subsect Selection technique prospers under this solving method due to the improved quality of the cores (as discussed in [10, Chapter 5]) and a potential increase in the number of cores. However, the Mixture technique (in combination with non-incremental SAT-solving, Core Rejection and a Core Generation Phase of 720 seconds) slightly deteriorates in performance when using this form of IHS. Contrarily, the Community technique sees a marginal improvement in its performance when utilising IHS with non-optimal hitting sets. A detailed analysis can be seen in Section 6.4

Finally, the accuracy of the translated lower bound between the various techniques differs significantly; with the Mixture technique boasting the largest discrepancy between the translated lower bound and the actual lower bound. It should be noted that if the threshold for Core Rejection is sufficiently low then the performance of the Mixture technique is expected to be similar to `RC2`. As expected, the Community technique generates an increased translated lower bound due to the

hypergraph splitting oftentimes rejecting fewer cores. Interestingly, the Subset Selection technique boasts the smallest difference between the translated lower bound and the actual lower bound which indicates that selecting disjoint cores does not necessarily introduce the "cost" of a lower translated bound. Notably, the discrepancy between the bounds when using IHS with non-optimal hitting sets is much higher for the Mixture and Community techniques than the Subset Selection technique which is potentially due to a difference in the generated core structures. These results, in terms of the accuracy of the translated lower bound, indicate that this factor is not necessarily the crucial factor influencing performance but that which cores are used for seeding the Final Solving Phase play a significant role in determining the hybrid algorithm's efficacy.

6.3. Detailed Results

Definition 14
A **cactus plot** is a plot ubiquitous in the (Max)SAT field, representing the overall manner in which a solver solves instances. This plot is created by determining all of the solving times for a set of instances and then plotting these times for the instances which are solved in ascending order. While disallowing a direct comparison between instances, it facilitates a straightforward interpretation of the trends in the performance of the solver. The further to the right in the plot a trace ends, the more instances it has solved.

Definition 15
The ratio between the translated lower bound tlb generated by the Subset Selection technique and the actual lower bound lb determined by the IHS-solver is denoted by $\boldsymbol{\alpha_T} = \frac{tlb}{lb}$ where T is one of the techniques specified in Section 6.1.

The difference between the translated lower bound and the actual lower bound is denoted by $\triangle_{\mathbf{T}} = lb - tlb$

6.3.1. Results Mixture

Figure 6.1 shows the cactus plot (see Definition 14) of the performance of the mixture algorithm in comparison to RC2.

Mixture

It can be seen from the plot that the original mixture algorithm (without any additional techniques) performs the worst of all the approaches, this is likely due to the lack of expressivity of the translated cores. Generally, the cores which are found by this algorithm are large which leads to a pseudo-linear search on a connected component. If the connected component encompasses a copious amount of variables then this is likely to impact the performance negatively. In total, this approach solves 322 instances.

Mixture$_{\texttt{CR}}$

The Core Rejection described in Section 5.4.5 in combination with the Mixture algorithm appears to significantly increase the performance (as can be seen when observing Mixture$_{CR}$), solving 354 instances versus the 322 of the "pure" Mixture algorithm. This indicates the potential in allowing the OLL-solver to create more complex constraints in certain cases rather than solely relying on the constraints found by the IHS-solver during the Core Generation Phase. To a certain extent, Core Rejection leads to the Mixture algorithm potentially performing similarly to `RC2`. It is a question of hyperparameter tuning in which scenarios it would be beneficial to reject cores as this is likely instance dependent.

Mixture$_{\texttt{CR+NI}}$

An interesting observation based on the plot is that disallowing the SAT-solver from learning clauses improves the performance rather significantly (as can be seen when observing Mixture$_{CR+NI}$), solving 366 instances compared to the 354 or 322 instances of the previous approaches. The

same incremental oracle being used during both the Core Generation Phase and the Final Solving Phase in an effort to reduce the space usage of the solver can thus potentially lead to decreased performance of the SAT-solver. This is likely due to the copious amount of learned clauses added during the Core Generation Phase hampering the performance in the Final Solving Phase.

Mixture$_{\text{CR+NI+720s}}$ & Mixture$_{\text{CR+NI+1800s}}$

The amount of time spent in the Core Generation Phase can have a pronounced influence on the number of instances solved. As can be seen in Figure 6.1, Mixture$_{CR+NI+720s}$ and Mixture$_{CR+NI+1800s}$ both solve 375 instances (which is four more than `RC2`). This indicates that there is a substantial amount of information to be gained when performing the Core Generation Phase over an extended period of time. The similarity in the number of instances solved by both of the aforementioned approaches suggests that there is a trade-off between the amount of information gained by generating cores and the amount of time provided for the OLL-solver to determine the optimal solution. It is thus paramount to provide sufficient time for the OLL-solver to solve to optimality.

An interesting observation is that both Mixture$_{CR+NI+720s}$ and Mixture$_{CR+NI+1800s}$ have an upwards-sloping trend at an earlier point in time than the other techniques. Notably, these steep increases lead to a plateau in the graph at approximately 720 seconds and 1800 seconds. This is likely due to the fact that the implementation of the algorithm used for the Core Generation Phase is suboptimal leading to middling instances being solved at a slower pace. However, as can be observed, this does not appear to introduce significant issues for more difficult instances, leading to adequate performance.

VBS$_{\text{Mixture}}$

VBS$_{Mixture}$ solves the most instances, solving 389 instances whereas `RC2` solves 371 instances and Mixture$_{CR+NI+720s}$ solves 375 instances. This provides the initial proof that there are contexts in which the Mixture algorithm can indeed improve the performance compared to the traditional OLL-solver `RC2`. This idea is further strengthened by the competitive performance of Mixture$_{CR+NI+720s}$ as the instances solved by this approach differ from the instances solved by `RC2` thus indicating the beneficial effect of the Core Generation Phase for certain instances.

Translated Lower Bound Accuracy

As can be observed in Table 6.1, when utilising no techniques the difference between the translated and actual lower bound (signified by $\triangle$ and α as defined in Definition 15) is minimal (the discrepancies can be explained by the interruption of the SAT/MIP-solver not being immediate). The other techniques have a similar $\triangle$ and α with the exception of the approaches utilising an extended Core Generation Phase which generally have an increased $\triangle$ but a similar α. This difference in $\triangle$ can be explained by the fact that a protracted Core Generation Phase generally leads to a higher lower bound, a comparable α thus means that these techniques start with a higher lower bound than their counterparts. Furthermore, this indicates that the amount of cores being rejected does not necessarily increase linearly in proportion to the amount of time spent in the Core Generation Phase.

Technique	**Average $\triangle_{\text{Technique}}$ (Solved - Unsolved)**	**Average $\alpha_{\text{Technique}}$ (Solved - Unsolved)**
No Technique	0.08 (0.09 - 0.07)	99% (98% - 99%)
CR	38.16 (25.25 - 50.32)	43% (35% - 50%)
CR+NI	39.98 (27.61 - 53.94)	42% (37% - 49%)
CR+NI+720s	60.32 (34.37 - 86.42)	42% (36% - 48%)
CR+NI+1800s	73.69 (37.82 - 105.54)	40% (31% - 47%)

Table 6.1: Metrics related to the lower bound for each of the Mixture techniques (with the addition of metrics split by whether an instance was solved or unsolved)

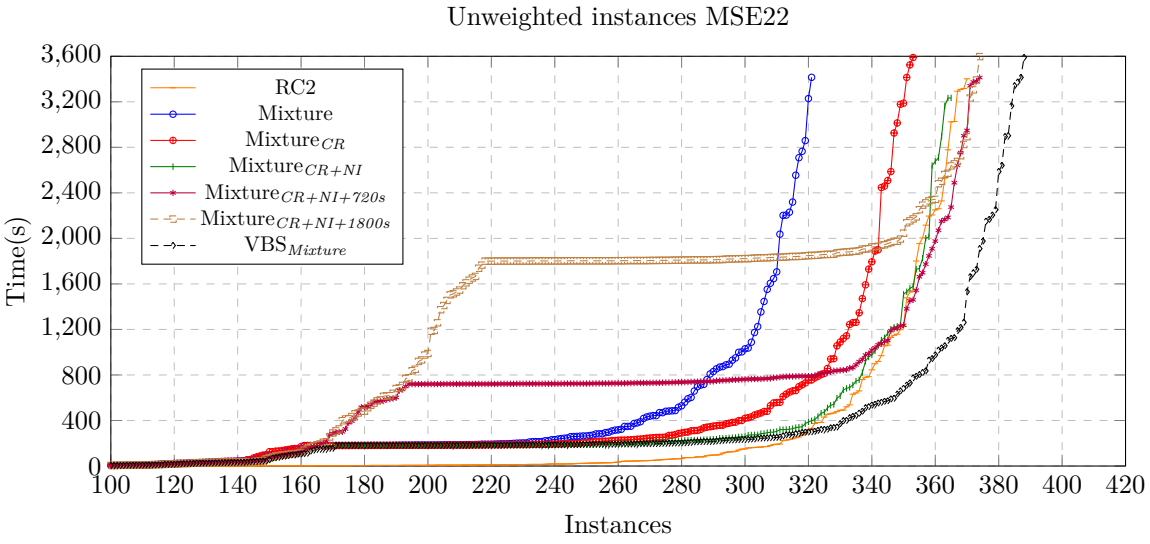


Figure 6.1: Cactus plot of Mixture approach for MSE22 instances

6.3.2. Results Community

Figure 6.2 shows the cactus plot of the Mixture algorithm in combination with the hypergraph cutting as discussed in Section 5.4.4 in comparison to `RC2`.

Community

As can be seen in the plot, the Community algorithm without any adjustments performs the worst out of all the different approaches in terms of the number of instances solved, solving a total of 335 instances. This indicates that there are numerous factors affecting the performance of this approach which interact with one another in non-obvious manners. It should be noted that this approach solves more instances than the Mixture approach as showcased in Section 6.3.1 while solving fewer instances than Mixture$_{CR}$ indicating that occasionally, the overhead generated by this approach is not beneficial to the overall search process.

Community$_{\texttt{1CT}}$

Interestingly, Community$_{1CT}$ performs better than the Community approach, solving 345 instances compared to the 335 of the previous approach. This shows that the threshold after which the hypergraph cutting is performed (in this case if the core is larger than 1% of the number of soft clauses) influences whether certain instances are solved. A lower threshold performing well demonstrates that for certain instances the default threshold is excessive, meaning that it is possible to benefit from the effects of hypergraph cutting even when the size of the connected component is not as significant.

Community$_{\texttt{NI}}$

Community$_{NI}$ once more exhibits that not using incrementality can be beneficial to the overall solving process. This approach solves 364 instances compared to the 335 instances and 345 instances of the aforementioned approaches. Similarly to the previous section, this indicates that the copious amount of clauses learned during the Core Generation Phase is impeding the performance of the SAT-solver in later stages.

Community$_{\texttt{NI+10CT}}$

The behaviour of Community$_{NI+10CT}$ is similar to that of Community$_{NI}$, solving 366 instances in total. It appears that the influence of the hypergraph splitting threshold is influential up to a certain point after which it does not deteriorate nor improve the performance substantially. One potential explanation is that there are not many instances where the size of the connected components is between 5% and 10% of the number of soft clauses. This would illustrate that the

connected components found by the IHS-solver during the Core Generation Phase are generally within a certain range with regard to size (which is context-dependent).

Community$_{\text{NI+0MHS}}$
Community$_{NI+0MHS}$ is interesting in terms of its performance. It does not make use of incrementality but in general, performs notably inferior to the other approaches which likewise do not make use of incrementality. In total, this approach solves 349 instances. This indicates that allowing connected components with a lower bound of zero is oftentimes detrimental to performance. The manner in which these components are encoded can lead to significantly more complex constraints by the exorbitant amount of variables added to the initial complex constraint. The intuition behind this detrimental effect is based on the fact that if the variables within a connected component are not present in any cores together then it is likely that these variables are not related to one another leading to poor performance when they are then assembled into a cardinality constraint.

Community$_{\text{NI+720s}}$ & Community$_{\text{NI+1800s}}$
The time spent in the Core Generation Phase can have a pronounced influence on the number of instances solved by Community. As can be seen in Figure 6.2, Community$_{NI+720s}$ solves 361 instances and Community$_{NI+1800s}$ solves 362 instances. The reason for this lack of performance compared to the results presented in Section 6.3.1 is the rising memory overhead introduced by the hypergraph clustering. As more cores are generated, more hyper-edges are added to the hypergraph which leads to increased memory consumption when the representation required for the clustering is created. This leads to numerous instances reaching out of memory errors owing to the additional memory required for creating the hypergraph representation used for the clustering. Interestingly, while using a Core Generation Phase of 1800 seconds introduces 9 more instances for which OOM errors occur compared to a Core Generation Phase of 720 seconds, the additional information provided by the protracted Core Generation Phase results in a single additional instance being solved.

VBS$_{\text{Community}}$
As expected, VBS$_{Community}$ solves the most instances, namely 399. Interestingly, the number of instances solved is higher than that of VBS$_{Mixture}$ of Section 6.3.1. This similarly indicates that there is potential for the mixture algorithm to use hypergraph splitting to enhance the performance of the solver. Furthermore, it demonstrates that there is a significant difference between which instances are solved by the different techniques.

Translated Lower Bound Accuracy
As can be observed in Table 6.2, a similar homogeneity of both $\triangle$ and α can be seen in this table as can be observed in Table 6.1. A notable exception is the translated lower bound of the $NI + 0MHS$ technique which is significantly closer to the actual lower bound than for the other techniques. This difference between $\triangle_{NI+0MHS}$ and $\triangle_{NI}$ exhibits that there are oftentimes cases in which the hypergraph splitting results in a component which contains no edges.

Furthermore, as opposed to the Mixture technique, an extended Core Generation Phase does not necessarily signify an increased $\triangle$. This could be due to two reasons, the first being that the hypergraph splitting adequately processes the increased number of cores. The second reason is that the instances which would result in a higher $\triangle$ run out of memory leading to these instances not being considered.

Technique	Average $\triangle_{\text{Technique}}$ (Solved - Unsolved)	Average $\alpha_{\text{Technique}}$ (Solved - Unsolved)
No Technique	0.39 (0.24 - 0.52)	94% (91% - 97%)
1CT	8.44 (5.08 - 12.14)	69% (64% - 75%)
NI+10CT	11.23 (7.86 - 15.7)	71% (68% - 75%)
NI	8.77 (6.17 - 12.34)	71% (68% - 76%)
NI+0MHS	0.36 (0.2 - 0.54)	97% (97% - 97%)
NI+720s	10.11 (7.99 - 12.87)	72% (66% - 79%)
NI+1800s	10.64 (9.23 - 12.45)	70% (62% - 79%)

Table 6.2: Metrics related to the lower bound for each of the Community techniques (with the addition of metrics split by whether an instance was solved or unsolved)

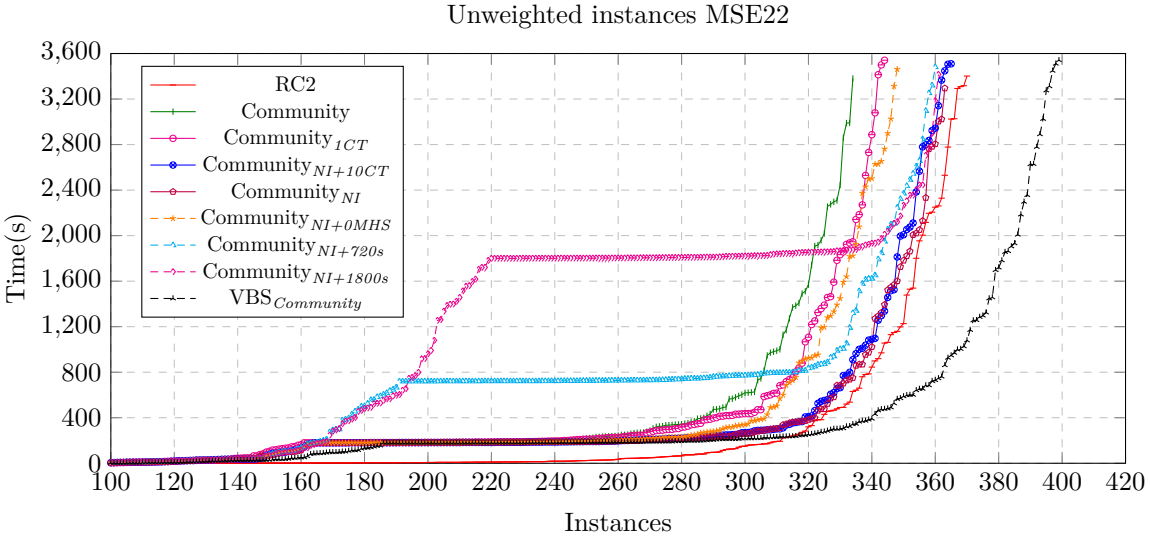


Figure 6.2: Cactus plot of Community approach for MSE22 instances

6.3.3. Results Subset Selection

Figure 6.3 shows the cactus plot of the Subset Selection technique specified in Section 5.5.

$\text{Subset}_{\text{MO}}$ & $\text{Subset}_{\text{SmS}}$

The performance of Subset_{MO} and Subset_{SmS} is comparable in terms of the number of instances solved, both solving 372 instances. Interestingly, there are five instances which are solely solved by Subset_{MO} and five instances which are solved solely by Subset_{SmS}. This indicates that there are certain instances for which the approaches result in different cores which are less or more suited to the instance's structure.

$\text{Subset}_{\text{TB}}$

The Time-based subset selection method shown in Subset_{TB} performs slightly substandard compared to the other methods, yet it nevertheless achieves an adequate number of instances solved, namely 367. The decreased number of instances solved could be explained by the lack of foresight of this method when selecting the cores. This selection method does not take into account in what manner the cores which are selected will influence the subsequently selected cores. In certain cases, this is likely to lead to cores being selected which do not contribute to the performance of the Final Solving Phase positively.

$\text{VBS}_{\text{Subset}}$

VBS_{Subset} solves the most instances, solving a total of 380. This once more indicates that for certain instances these techniques enhance the performance when compared to `RC2`. Interestingly,

the Subset Selection techniques boast similar performance to its counterparts based on the hypergraph. This indicates that this technique can still find the correct "building blocks" despite disregarding numerous cores during its core selection process. Additionally, the performance of the VBS compared to the individual techniques does not seem to differ significantly. This indicates that there is a significant amount of overlap between the instances solved by the different Subset Selection techniques.

Translated Lower Bound Accuracy

While Subset_{SmS} solves the same number of instances as Subset_{MO}, it nevertheless boasts a larger $\triangle_{SmS}$. The average difference for this strategy is 9.39 and α_{SmS} is equal to 83%. This further supports the hypothesis that the accuracy of the translated lower bound is not necessarily the quintessential element of the hybrid algorithm but that oftentimes the choice of which cores to translate is of the utmost importance.

Furthermore, as can be observed in Table 6.3, $\triangle_{TB}$ is the largest when compared to the other techniques. However, the value of α_{TB} is greater than that of Subset_{SmS}. This indicates that, for certain instances, the difference is much higher while percentage-wise the translated lower bound is more accurate. The subpar performance of this technique could be explained by it producing a significantly lower translated bound compared to the actual lower bound (resulting in the higher $\triangle_{TB}$) for certain instances which leads to inferior performance when juxtaposed with its counterparts.

Technique	Average $\triangle_{\text{Technique}}$ (Solved - Unsolved)	Average $\alpha_{\text{Technique}}$ (Solved - Unsolved)
MO	7.34 (6.5 - 8.31)	86% (85% - 87%)
SmS	9.39 (7.1 - 11.96)	83% (82% - 84%)
TB	10.14 (6.83 - 13.64)	85% (83% - 86%)
MO+NO	13.55 (10.53 - 17.62)	80% (81% - 79%)

Table 6.3: Metrics related to the lower bound for each of the Subset Selection techniques (with the addition of metrics split by whether an instance was solved or unsolved)

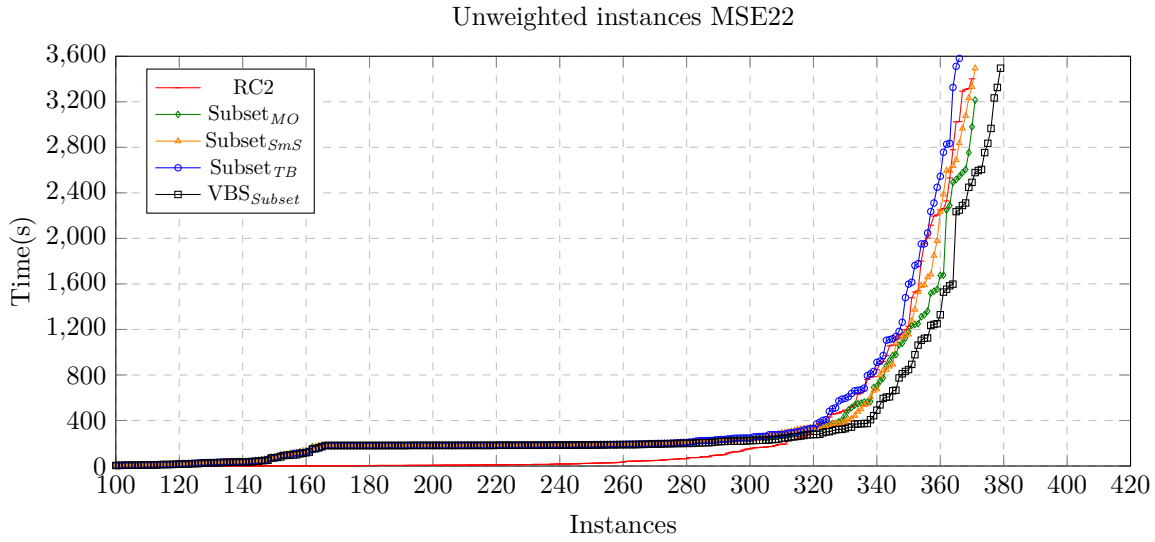


Figure 6.3: Cactus plot of Subset Selection approach for MSE22 instances

6.3.4. Technique Comparison

Figure 6.4 shows the performance of the techniques which solved the most instances as described in Section 6.3.1, Section 6.3.2 and Section 6.3.3.

It can be seen from the graph that the performance achieved by the solvers is similar in terms of the number of instances solved. Both Subset_{MO} and $\text{Mixture}_{CR+NI+720s}$ outperform `RC2` by

solving one more instance and four more instances, respectively. This similarity in performance indicates that the Core Generation Phase in general does not diminish the performance of the Final Solving Phase. However, the additional overhead introduced by Community$_{NI+10CT}$ is in certain instances not beneficial, likely due to the increased number of out-of-memory errors.

On certain instances, it can be observed that the Core Generation Phase indeed improves the performance quite substantially. This fact becomes readily apparent when analysing the results of VBS$_{ALL}$ which solves 404 instances compared to the 375 of the best-performing individual solver.

Translated Lower Bound Accuracy

As can be observed in Table 6.4, there is a significant discrepancy between the different techniques in terms of the translated lower bound and the actual lower bound found by the IHS-solver. Interestingly, it can be seen that the smallest $\triangle$ is that of Subset$_{MO}$ indicating that this technique (perhaps contrary to expectations) generates a higher translated lower bound than its counterparts. The main reason for this is the relatively low threshold of Core Rejection resulting in many cores being discarded for the other techniques. The technique with the second lowest $\triangle$ is Community$_{NI+10CT}$ which indicates that the hypergraph splitting does result in a higher number of cores being accepted (note that core rejection is not performed on components after they are split) and that this technique does not hamper performance to an intolerable extent. Finally, the technique which boasts the highest $\triangle$ by a significant margin is Mixture$_{CR+NI+720s}$. The general performance of this technique indicates that `RC2` can perform well on its own but that there are certain instances for which this technique aids in its performance. The fact that all of these techniques have similar performance further strengthens the belief that a higher α is not necessarily the paramount factor influencing performance but that the applicability of the techniques is instance dependent.

Technique	**Average $\triangle_{\text{Technique}}$ (Solved - Unsolved)**	**Average $\alpha_{\text{Technique}}$ (Solved - Unsolved)**
Mixture$_{CR+NI+720s}$	60.32 (34.37 - 86.42)	42% (36% - 48%)
Community$_{NI+10CT}$	11.23 (7.86 - 15.7)	71% (68% - 75%)
Subset$_{MO}$	7.34 (6.5 - 8.31)	86% (85% - 87%)

Table 6.4: Metrics related to the lower bound for each of the techniques in the comparison (with the addition of metrics split by whether an instance was solved or unsolved)

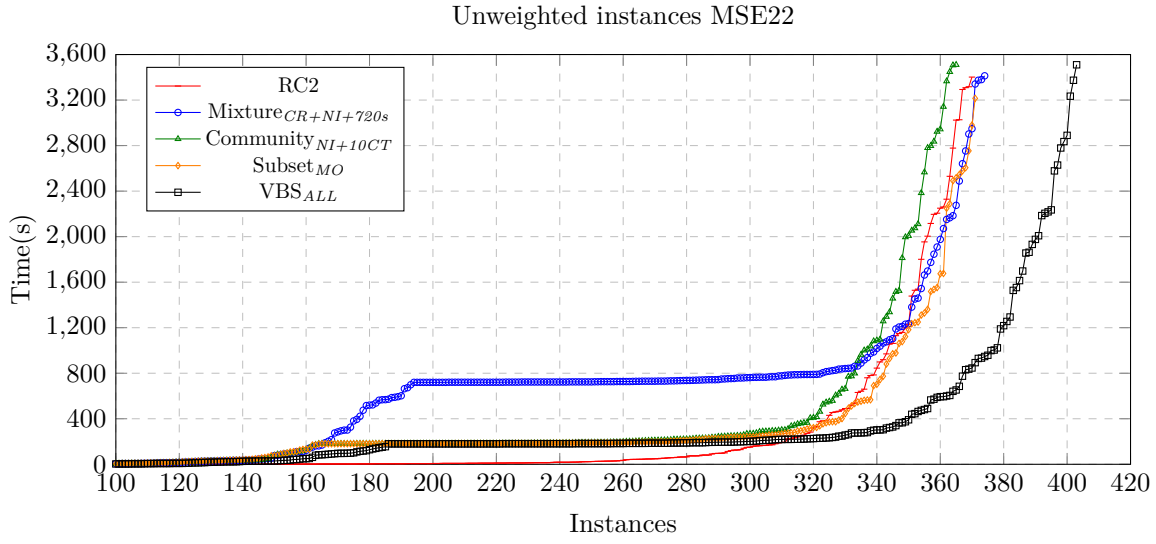


Figure 6.4: Cactus plot of different approaches for MSE22 instances

6.4. Results IHS with non-optimal hitting sets

Figure 6.5 shows the overview of the number of instances solved by the best technique for each approach using IHS with non-optimal hitting sets.

6.4.1. $\text{Subset}_{\text{MO+NO}}$

The most performant technique is Subset_{MO+NO}, solving a total of 385 instances compared to the 371 instances solved by RC2. This outstanding performance is likely caused by the inherent manner in which IHS with non-optimal hitting sets generates cores. As stated in [10, Section 5.4], more time is spent in the SAT-solver rather than in the MHS-solver. This leads to the conclusion that it is likely that more cores are generated using this form of IHS compared to "regular" IHS. Additionally, the manner in which the cores are generated influences the structure and variety of the found cores. In turn, these properties of IHS with non-optimal hitting sets provide the Subset Selection strategy with an ample choice of cores in addition to higher quality cores. For example, for the instance "cnf.15.p.6.wcnf.xz", both IHS approaches find a lower bound of eight. However, the subset selection using "regular" IHS starts the Final Solving Phase with a lower bound of six while the subset selection using non-optimal hitting sets starts with a lower bound of eight.

6.4.2. $\text{Community}_{\text{NI+NO}}$

Community_{NI+NO} boasts a similar performance to RC2, solving a total of 374 instances. A similar argument could be made as for the previous technique that the increase in performance is likely due to a higher lower bound caused by less time being spent in the MHS-solver. However, due to more cores being generated, the same memory issues mentioned in Section 6.3.2 occur, leading to fewer instances being solved. Nevertheless, more cores being generated could lead to better separability of the different connected components as either there are more options as to where to cut the connected component or the additional cores indicate that the connected component should not be separated thus preventing unnecessary overhead at a later stage in the solving process.

6.4.3. $\text{Mixture}_{\text{NI+CR+NO+720s}}$

$\text{Mixture}_{NI+CR+NO+720s}$ solves a similar number of instances as the previous technique, namely 373 instances. While still solving more instances than RC2, it solves the least instances out of all the techniques. This is likely due to the fact that the issues ameliorated by hypergraph splitting are not addressed leading to a decrease in performance. The potential additional cores thus lead to this technique creating larger constraints which are either rejected (potentially inducing extraneous strain in subsequent steps) or provided directly to the OLL-solver.

6.4.4. VBS_{NO}

Finally, VBS_{NO} exhibits the performance of the most suitable technique for each instance using IHS with non-optimal hitting sets. In total, the virtual best solver solves 411 instances which is substantially more than RC2. Compared to the VBS presented in Section 6.3.4, it solves 7 more instances indicating that IHS with non-optimal hitting sets is more suited to the Core Generation Phase than "regular" IHS. It should be noted that the number of instances solved by a VBS consisting of both versions of IHS is 419 which shows that both IHS approaches have particular instances on which they excel.

6.4.5. Translated Lower Bound Accuracy

Generally, the α of the techniques when using IHS with non-optimal hitting sets is overall lower than those of their "regular" counterparts. An explanation for this could be that a larger number of cores is generated which leads to more connected components being rejected due to the increase in size. An explanation as to why particularly Community_{NI+NO} has a lower α than its counterpart is that the structure of the cores found by IHS using non-optimal hitting sets does not accommodate the hypergraph splitting leading to more cores being rejected. Interestingly, the relative increase in $\triangle_{\text{Community}_{NI+NO}}$ when solely considering the unsolved instances compared to the $\triangle_{\text{Community}_{NI+NO}}$ when only considering the solved instances is much larger when IHS with non-optimal hitting sets is utilised.

Technique	Average $\triangle_{\text{Technique}}$ (Solved - Unsolved)	Average $\alpha_{\text{Technique}}$ (Solved - Unsolved)
$\text{Mixture}_{NI+CR+NO+720s}$	233.54 (83.37 - 372.92)	32% (32% - 31%)
Community_{NI+NO}	137.46 (42.54 - 246.57)	36% (37% - 35%)
Subset_{MO+NO}	13.55 (10.53 - 17.62)	80% (81% - 79%)

Table 6.5: Metrics related to the lower bound for each of the techniques in the comparison using IHS with non-optimal hitting sets (with the addition of metrics split by whether an instance was solved or unsolved)

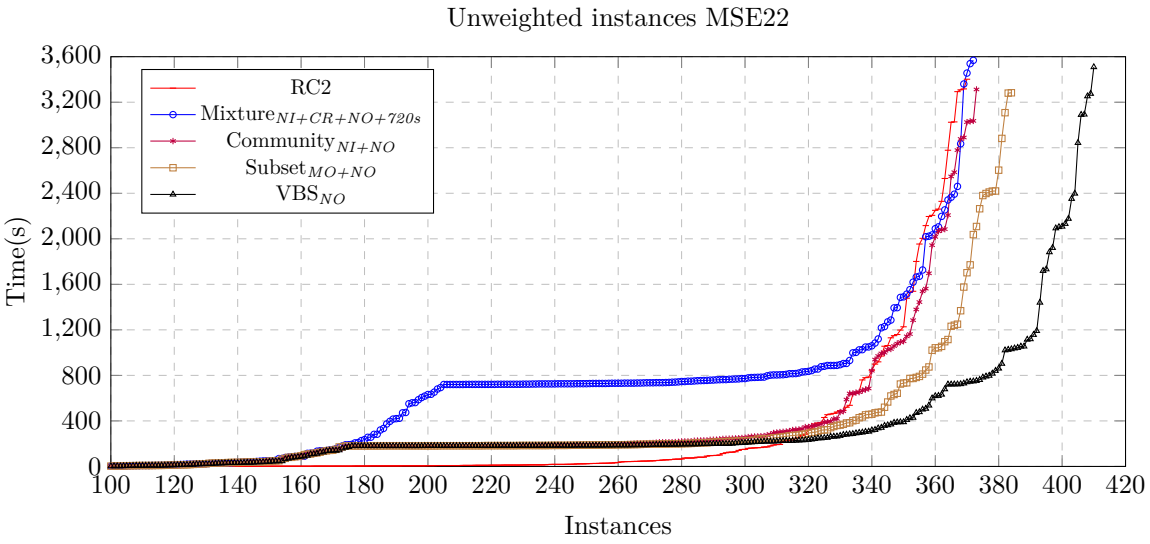


Figure 6.5: Cactus plot of different approaches using IHS with non-optimal hitting sets for MSE22 instances

6.5. Instance Analysis

This section will detail the analysis of the instances based on the results of Section 6.3. Furthermore, a portfolio approach is proposed utilising the findings of the aforementioned analysis.

6.5.1. Analysis Summary

Generally, the different techniques perform especially well on disparate instances, each instance being characterised by a number of features related to the number of cores, size of the cores and other metrics appertaining to the clauses themselves (e.g. the number of hard/soft clauses and the number of variables).

For the Subset Selection technique, it can be observed that this technique generally excels on instances with either somewhat smaller cores or cores of an inordinately significant size due to the fact that the smaller cores oftentimes allow the Subset Selection technique to determine a favourable lower bound and the other techniques are unable to ameliorate the issues inherent to sizeable cores (e.g. hypergraph splitting on a connected component consisting of substantial cores oftentimes leads to subpar OLL-core structures). A substantial number of small cores is thus likely to generate a lower bound using this technique which is close to the lower bound found by IHS.

The Community Technique, similarly to the Subset Selection technique, excels on instances which consist of a significant number of smaller cores. Intuitively this accords with the notion that it is simpler to perform clustering when the cores are small due to the fact that removing a single core oftentimes does not mar the overall structure. The difference between the instances solved by this technique and the Subset Selection technique appears to lie in the fact that the instances solved solely by the Community technique are comparatively dense. This denseness of the hypergraph of cores leads to improved separability of connected components as it is less likely that the resulting components after splitting contain no hyper-edges.

Furthermore, there is a single instance which is solved solely by the Mixture technique. The features of this instance related to the cores of the instance are in between the features of the

instances solely solved by the other two techniques. This indicates that this instance does not benefit from either of the other two techniques due to the fact that it neither allows the hypergraph splitting to perform well due to the relatively large core size nor does it allow the subset selection to find the right building blocks for creating the OLL-cores.

Moreover, when analysing the behaviour of the hybrid algorithm utilising IHS with non-optimal hitting sets, it can be observed that the superlative performance of the Subset Selection technique causes the number of instances solely solved by the other techniques to be insignificant. However, for the instances which are solely solved by the Subsect Selection technique, the same insights hold as mentioned in the previous paragraph(s), indicating that this technique prospers when instances boast either minute cores or sizeable cores.

Finally, an analysis of the number of instances solved by these techniques per benchmark family shows that there are certain families of instances which boast improved performance for different techniques. The families for which the largest differences can be seen oftentimes lead to analogous conclusions as mentioned previously (e.g. the Community technique does not perform as well for instances with large cores). When observing the behaviour of the different techniques when using IHS with non-optimal hitting sets, it can be shown that there are certain techniques which boast enhanced performance due to improved lower bounds.

6.5.2. Detailed Analysis

Core Data Gathering

The core data used in this section has graciously been provided by Nina Narodytska. The data is the result of running a core generation algorithm based on non-optimal hitting sets for a duration of six hours with randomization to ensure that the number of duplicate cores found during this process is minimized.

Overview

The main interest lies in the instances which are solved by a certain technique while not being solved by other techniques. The metrics which will be analysed in this section are the distribution of the cores, the average length of the cores and the number of cores generated.

The first part of the analysis will concern the best techniques of the hybrid algorithm with default values for the core generation phase (a duration of 180 seconds, non-incremental SAT solving and "regular" IHS). The best combinations of techniques for these categories are Mixture$_{CR+NI}$, Community$_{NI+10CT}$ and Subset$_{MO}$. There is a single instance which is solely solved by Mixture$_{CR+NI}$, seven instances which are solely solved by Community$_{NI+10CT}$ and six instances which are solely solved by Subset$_{MO}$. Figure 6.6 shows an overview of the metrics related to the core size and number of cores per technique. The remainder of the section will focus on discussing these metrics.

Core size

Figure 6.6a shows the probability density function (PDF)[2] of all of the cores across the instances solved by a single technique.

Interestingly, it can be observed that the technique Community$_{NI+10CT}$ (aptly named CommunityNI10CT in the plot) generally solves instances which have a low number of cores. This trend can also be observed when examining the average core size in Figure 6.6d which is significantly lower compared to the other two techniques. It is postulated that the superlative performance of this technique is caused by the relatedness of the cores, thus facilitating a beneficial influence of the hypergraph cutting technique. Furthermore, small cores enhance the flexibility of the hypergraph cutting as removing a small hyper-edge from the graph oftentimes leaves the cores within the resulting connected components intact.

Furthermore, Figure 6.6a exhibits that the instances which benefit from the technique Subset$_{MO}$ (aptly named SubsetMO in the plot) are more spread out in terms of the core size compared to the other techniques. It can be observed that there are two significant groups of bins, the first being at the lower end centred around a core size of ten indicating that there are certain instances with

[2] A PDF is used rather than a histogram due to the large relative difference between the number of cores

a low core size which benefit from subset selection, the second being around a core size of 150-200. For the latter group of bins, it is likely of the utmost importance to determine the right building blocks to avoid the superfluous complex constraints introduced by the OLL-solver. Judging by Figure 6.6b, there are two instances for which the average core size is indeed quite large, providing support for the previous hypothesis. As for the cores related to the other three instances, it can be seen in Figure 6.6b that generally, the cores in these instances are relatively small but that there are a significant number of cores. This could indicate that retaining a satisfactory lower bound is possible due to a (potentially) small amount of overlap between the different cores.

Moreover, the single instance which is solely solved by Mixture$_{CR+NI}$ (aptly named Mixture-CrNoIncr in the plot) can be observed to have a varying number of cores when compared to the other instances which are generally clustered around the lower end of the plot and a core size of between 150-200. The cores for this technique range from 1 to approximately 125 with an average core size of roughly 45. The number of cores for this instance is relatively low compared to others but there are instances which can be observed in Figure 6.6b which have a similar (or slightly lower) number of cores. There is likely a combination of factors which cause this instance to benefit from neither the hypergraph splitting nor subset selection techniques. One factor is the number of soft clauses which influences the thresholds related to core rejection and hypergraph cutting. The number of soft clauses for this particular instance is approximately 2500 (5% of which is 125 and 10% of which is 250). The relatively low thresholds related to the number of soft clauses in combination with the relatively large average size could indicate that hypergraph splitting is likely to occur frequently thus introducing a certain amount of overhead which is likely not beneficial due to the difficulty in clustering the graph. When analysing the translated cores it can be observed that Mixture$_{CR+NI}$ starts the Final Solving Phase with a lower bound of 28, consisting of solely single variable cores, while Community$_{NI+10CT}$ starts with a lower bound of 63 where the hypergraph cutting is applied to a single connected component. However, the Final Solving Phase of Community$_{NI+10CT}$ does not raise the bound from 63 indicating that the resulting SAT instance after translating the cores is exorbitantly difficult to solve. Finally, Subset$_{MO}$ starts with a lower bound of 57 with a number of the translated cores being of substantial size. Interestingly, this technique finds a lower bound equal to the optimal solution cost of 65 but it does not prove optimality within the time limit, indicating that the correct building blocks were not present for `RC2` to efficiently find the solution. This particular instance shows certain drawbacks of the aforementioned techniques based on the core structure of the instances, showing that in certain instances not convoluting the solution space provides superior performance.

Number of cores

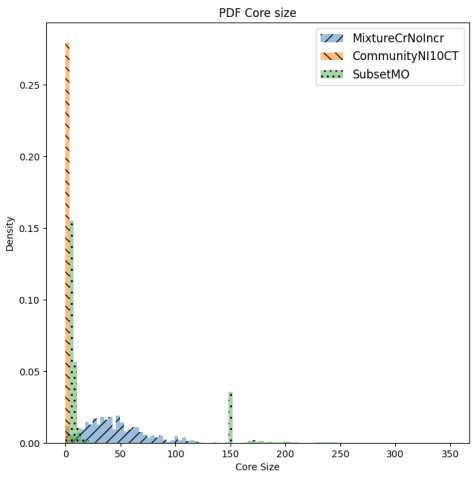
Figure 6.6c shows a histogram of the number of cores for the different instances which have been partially discussed in the previous section.

As can be seen in Figure 6.6d, the average number of cores for instances solely solved by Community$_{NI+10CT}$ is approximately half that of the instances solved by Subset$_{MO}$. This could indicate that the instances are easy to solve for the SAT-solver, however, such a generalisation is likely to be unsound as the ease with which a solver can find cores of the original formula is not invariably indicative of the complexity of the MaxSAT instance. However, the number of cores of an instance can be suggestive of an increase/decrease in certain graph metrics such as the diameter and density of the hypergraph of the cores (an overview of such metrics can be seen in [41]) which could influence the performance. It can be observed in Figure 6.6d that the average density across the instances solely solved by Community$_{NI+10CT}$ is indeed higher than those of the other instances, particularly the instance "navigation_5x5_10.wcnf.xz" is well-connected. While the other instances are less connected it could be the case that there are certain clusters of well-connected connected components which allow the hypergraph cutting technique to prosper.

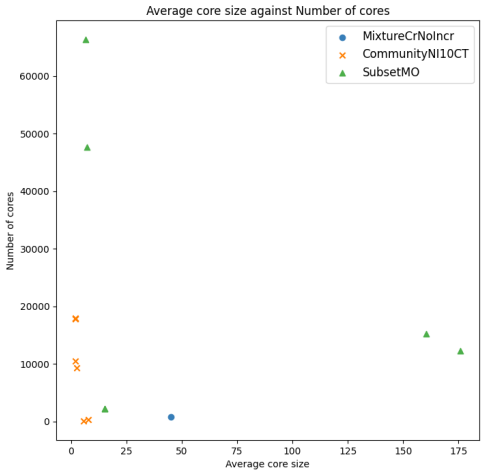
The average number of cores for instances solved solely by Subset$_{MO}$ is the highest of all the techniques. This could indicate that it is likely that the lower bound found by the subset selection strategy will be higher than for other instances due to a high number of small cores having a lower chance of overlapping than cores of a larger size. It is thus expected that in these scenarios the

subset selection strategy will flourish.

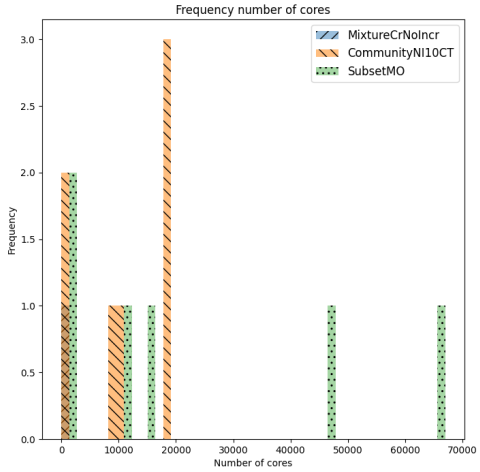
Finally, the average number of cores for the single instance solved by Mixture$_{CR+NI}$ is low compared to the average core size of the other techniques. However, as can be seen in Figure 6.6c, there are other instances which have a similar size. Furthermore, the average density of the graph is rather low compared to other instances indicating that the graph is not well-connected. This combination of a low connectedness and a low number of cores with generally higher sizes prevents the other techniques from performing well while this does not limit the performance of Mixture$_{CR+NI}$.



(a) PDF of the core sizes of the instances solely solved by a single technique using "regular" IHS



(b) Core size against the Number of cores of the instances solely solved by a single technique using "regular" IHS



(c) PDF of the number of cores of the instances solely solved by a single technique using "regular" IHS

Technique	Average number of cores	Average core size	Estimated Average Density
MixtureCrNoIncr	791	45.1	0.012
CommunityNI10CT	10568.4	3.4	0.154
SubsetMO	24343.7	63.3	0.039

(d) Average metrics per technique using "regular" IHS

Figure 6.6: Metrics related to cores for instances only solved by a single technique using "regular" IHS

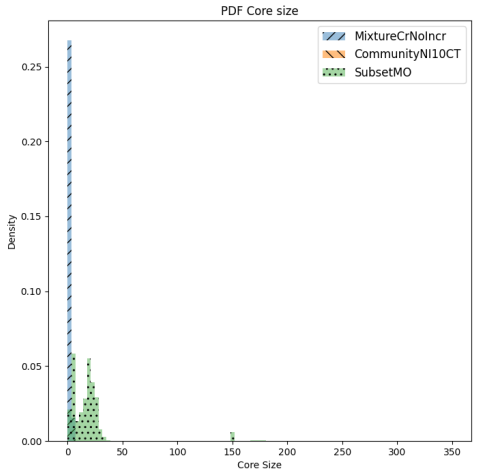
IHS with non-optimal hitting sets

The instances which are solved solely by a particular technique described in Section 6.4 when using IHS with non-optimal hitting sets show a somewhat different trend from the instances discussed in Section 6.5.2. The superlative performance of the Subset Selection technique causes this particular

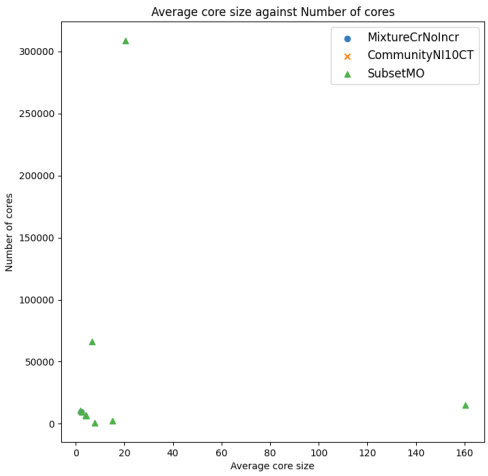
technique to solve a total of 15 instances which are not solved by any other technique while the Mixture technique only solves a single instance which is not solved by any other technique. There is not a single instance which is solely solved by the Community approach, indicating that the Subset Selection approach excels to such an extent that all instances solved by Community are likely to be solved by this technique as well.

As can be seen in Figure 6.7, the distribution of instances solely solved by Subset_{MO} (in terms of the number of cores and the core size) is similar to the distribution presented in Section 6.5.2 in the sense that there are several instances clustered around the lower end of the plot while there is also an instance for which the number of cores is low but the core size is quite substantial. It should be noted that there is a certain amount of overlap in terms of the instances which are solved by Subset_{MO} between "regular" IHS and IHS with non-optimal hitting sets. This illustrates that the difference in core structure between instances is present, nevertheless Subset_{MO} is still able to discover a "good" core structure to translate. However, the average core size shown in Figure 6.7d is substantially lower than the average core size presented in Figure 6.6d which suggests that this technique benefits from the (potential) increase in the number of cores specifically if the average core size is small. However, it is also possible that the other techniques benefit from the different cores found by IHS with non-optimal hitting sets leading to certain instances with higher average core sizes being solved by other techniques as well.

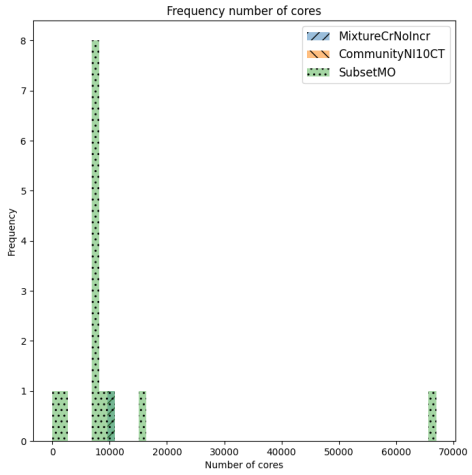
The differences between the metrics of the instance solely solved by Mixture using both versions of IHS are quite substantial, particularly the average core size which is 2.4 for IHS with non-optimal IHS and 45.1 for "regular" IHS. It should be noted that these metrics are solely based on a singular instance in both cases and thus the conclusions drawn are not generalisable. Furthermore, the instance solely solved by Mixture presented in Section 6.5.2 ("normalized-ex5.pi.opb.msat.wcnf.xz") is also solved when using IHS with non-optimal hitting sets while the reverse is not true for the sole instance presented in this section ("MinFill_R0_mulsol.i.4.wcnf.xz"). This exhibits the general unpredictability of the hybrid algorithm and the importance of the structure of the discovered cores.



(a) PDF of the core sizes of the instances solely solved by a single technique using IHS with non-optimal hitting sets



(b) Core size against the Number of cores of the instances solely solved by a single technique using IHS with non-optimal hitting sets



(c) PDF of the number of cores of the instances solely solved by a single technique using IHS with non-optimal hitting sets

Technique	Average number of cores	Average core size	Estimated Average Density
MixtureCrNoIncr	9626	2.4	0.007
CommunityNI	-	-	-
SubsetMO	31259.4	16.6	0.055

(d) Average metrics per technique using IHS with non-optimal hitting sets

Figure 6.7: Metrics related to cores for instances only solved by a single technique using IHS with non-optimal hitting sets

Analysis per benchmark family

Table 6.6 exhibits the results per (estimated) benchmark family (the instance families used in this work are presented in Appendix A).

The first notable observation can be made for the ”des” instance family, consisting of 16 instances that represent Discrete-Event System (DES) Diagnosis benchmarks[3]. These instances boast a sizeable average core length of approximately 350 literals. As discussed in the previous section(s), the Community approach potentially possesses subpar performance on instances with substantial cores due to the limited applicability of hypergraph splitting. The other techniques perform reasonably well while still being outperformed by `RC2`. This could be explained by a number of cores being rejected which discards a sufficient amount of information resulting in subpar

[3]http://www.maxsat.udl.cat/13/solvers/README

performance.

However, the opposite conclusion can be drawn for the ”frb” instance family, also consisting of 16 instances that represent numerous Max-2-SAT[4] instances. The construction of these instances ensures that the MUSes are all of size 2 while the number of cores varies depending on the number of hard clauses. The superlative performance of the Community technique can be explained by the small size of the cores which allow for improved separability. An example of this can be seen in the instance ”frb30-15-5.partial.wcnf” for which the Mixture technique translates a lower bound of 93 while Community retains the same lower bound as found by the IHS-solver of 308 by splitting a single connected component into four subcomponents. The discrepancy between the lower bound of the original component and the resulting split component is relatively minute which indicates that the different components can be separated efficiently. It should be noted that in this case, the Subset Selection technique struggles to select cores due to the exorbitant cost of calculating the minimum overlap.

While there are more instance families for which there are discrepancies, these dissimilarities are oftentimes not substantial enough to draw generalizable conclusions with regard to the applicability of the techniques. Furthermore, there are families of instances for which none of the techniques performs adequately, such as ”decision-tree”, ”maxcut” and ”set-covering”. It should be noted that the solvers used within this work do not utilise many of the improvements as suggested in [10] and [6] which could explain the lack of performance on these instances. Another issue which could inhibit the performance is the fact that the algorithms do not take into account that instances which contain duplicate soft clauses could be converted to weighted instances.

[4] https://web.archive.org/web/20170716215620/http://www.nlsde.buaa.edu.cn/kexu/benchmarks/max-sat-benchmarks.htm

Instance Family (# of instances)	RC2	Mixture$_{\text{CR+NI}}$	Community$_{\text{NI+10CT}}$	Subset$_{\text{MO}}$
aes-key-recovery (16)	16 (80.83)	16 (84.67)	16 (37.6)	16 (66.44)
aes (7)	1 (0.15)	1 (10.32)	**2 (120.67)**	**2 (107.89)**
atcoss (13)	5 (190.68)	5 (197.93)	**6 (643.16)**	5 (206.82)
bcp (16)	16 (0.18)	16 (22.81)	16 (22.78)	16 (22.71)
close_solutions (16)	16 (34.13)	16 (156.22)	16 (155.86)	16 (152.92)
decision-tree (13)	0 (-)	0 (-)	0 (-)	0 (-)
des (16)	**16 (458.86)**	13 (635.96)	8 (516.74)	13 (561.92)
drmx-atmost (16)	16 (20.61)	16 (207.03)	16 (198.24)	16 (220.1)
drmx-cryptogen (16)	0 (-)	0 (-)	0 (-)	0 (-)
exploits-synthesis_changjian_zhang (3)	2 (1231.87)	2 (521.34)	2 (1298.39)	2 (1273.62)
extension-enforcement (16)	**8 (10.31)**	6 (163.04)	6 (174.69)	**8 (160.44)**
fault-diagnosis (16)	**16 (240.97)**	15 (85.29)	15 (84.46)	**16 (100.48)**
frb (16)	7 (4.4)	8 (182.4)	**12 (214.71)**	7 (127.12)
gen-hyper-tw (16)	3 (195.45)	3 (90.47)	3 (87.2)	3 (88.75)
HaplotypeAssembly (6)	6 (22.99)	6 (84.36)	6 (65.49)	6 (68.74)
hs-timetabling (1)	0 (-)	0 (-)	0 (-)	0 (-)
kbtree (14)	1 (0.04)	1 (4.36)	1 (4.25)	1 (4.18)
large-graph-community-detection_jabbour (8)	5 (0.01)	5 (4.87)	5 (4.81)	5 (4.68)
logic-synthesis (16)	10 (25.35)	**15 (156.46)**	13 (36.81)	14 (49.08)
maxclique (16)	13 (19.43)	13 (168.46)	13 (170.87)	13 (169.59)
maxcut (14)	0 (-)	0 (-)	0 (-)	0 (-)
MaximumCommonSub-GraphExtraction (16)	14 (182.42)	15 (460.71)	**16 (663.99)**	**16 (578.96)**
MaxSATQueriesInterpretableClassifiers (16)	9 (0.18)	9 (41.66)	9 (41.78)	9 (41.51)
mbd (16)	16 (3.55)	16 (168.56)	16 (167.51)	16 (164.62)
min-fill (16)	**5 (152.92)**	4 (4.17)	**5 (63.4)**	**5 (524.42)**
optic (16)	4 (1.77)	4 (161.73)	2 (137.71)	**6 (422.35)**
phylogenetic-trees_berg (15)	**14 (657.21)**	**14 (646.65)**	13 (638.66)	**14 (652.09)**
planning-bnn (15)	**12 (1537.81)**	10 (1070.03)	**12 (1445.82)**	9 (740.09)
program_disambiguation-Ramos (10)	8 (268.76)	8 (307.34)	8 (268.06)	8 (258.75)
protein_ins (12)	12 (123.26)	12 (257.34)	12 (292.22)	12 (254.38)
railroad_reisch (11)	0 (-)	0 (-)	0 (-)	0 (-)
railway-transport (6)	2 (9.73)	2 (106.14)	2 (106.55)	2 (106.95)
ramsey (2)	0 (-)	0 (-)	0 (-)	0 (-)
RBAC_marco.mori (16)	16 (12.86)	16 (88.38)	16 (88.16)	16 (85.77)
scheduling (5)	**2 (1267.19)**	1 (244.13)	**2 (842.45)**	**2 (1608.65)**
scheduling_xiaojuan (16)	5 (405.98)	5 (341.27)	5 (850.64)	5 (343.94)
SeanSafarpour (16)	**14 (171.39)**	12 (124.21)	12 (124.07)	12 (123.96)
security-witness_paxian (16)	16 (454.99)	16 (633.45)	16 (622.17)	16 (463.52)
set-covering (16)	0 (-)	0 (-)	0 (-)	0 (-)
setcover-rail_zhendong (4)	0 (-)	0 (-)	0 (-)	0 (-)
treewidth-computation (16)	13 (215.97)	**14 (386.71)**	**14 (333.69)**	**14 (342.54)**
uaq (16)	**13 (219.37)**	**13 (441.33)**	12 (540.79)	**13 (377.36)**
uaq_gazzarata (11)	**9 (235.22)**	8 (24.29)	8 (24.22)	8 (24.31)
xai-mindset2 (17)	11 (131.96)	11 (197.94)	11 (265.94)	11 (245.23)
reversi (5)	2 (0.25)	2 (16.14)	2 (15.04)	2 (13.85)
spinglass (2)	1 (7.36)	1 (184.2)	1 (184.05)	1 (184.43)
causal (16)	16 (133.39)	16 (243.02)	16 (269.71)	16 (219.72)

Table 6.6: Number of instances solved per benchmark family by the different techniques (average solving time in seconds between brackets)

IHS with non-optimal hitting sets

Table 6.7 exhibits the results per (estimated) benchmark family when using IHS with non-optimal hitting sets.

When comparing the results of this table with the results presented in Table 6.6, certain discrepancies become apparent. When observing the "des" instance family, the differences between the techniques are now negligible when compared to its "regular" IHS counterpart. Interestingly, the translated lower bounds of the various techniques are relatively low due to the nature of the core structure. However, rather than timing out as was the case previously, it proceeds to the Final Solving Phase and finds the correct solution.

Another interesting observation can be made about the "drmx-cryptogen" instance family. Using "regular" IHS, none of these instances could be solved by any of the techniques, however, when using IHS with non-optimal hitting sets, the Subset Selection technique solves half of the instances. The instances in this family can be characterised by a high optimal solution cost (oftentimes larger than 800). An analysis of the lower bounds found by the different techniques indicates that for the eight instances which are solved by IHS with non-optimal hitting sets that this form of IHS finds a much higher lower bound than its counterpart. While not solving to optimality, this form of IHS does find the optimal solution cost after which the Subset Selection technique selects numerous disjoint cores to provide to the Final Solving Phase. The incapability of the other techniques to solve these instances can be explained by the Core Rejection technique which prunes an excessive amount of cores from translation leading to subpar performance. This exhibits that for certain instance families, a prominent factor which influences performance is the choice of IHS algorithm which is utilised during the Core Generation Phase in combination with the thresholds at which other techniques are applied.

While the Community technique solves approximately seven more instances in the "des" instance family, it solves 5 fewer instances in the "frb" instance family. The cause of this could, once again, be related to the threshold of the Core Rejection. For certain instances within this instance family, the cores found using "regular" IHS can be split into subcomponents while this is not the case for the cores found using IHS with non-optimal hitting sets. This means that the components which are not split properly using the latter approach are likely to be rejected due to their excessive size leading to similar performance as the other techniques.

For the remainder of the instance families, the performance when using "regular" IHS and IHS using non-optimal hitting sets is comparable. For every technique analysed in this section, the usage of IHS with non-optimal hitting sets improves performance (it should be noted that this is not the case for all techniques, as can be seen in Section 6.4). The main factor contributing to this increase appears to be the difference in the lower bound between the two variations of IHS.

Instance Family (# of instances)	RC2	Mixture$_{\text{CR+NI+NO}}$	Community$_{\text{NI+10CT+NO}}$	Subset$_{\text{MO+NO}}$
aes-key-recovery (16)	16 (80.83)	16 (108.79)	16 (103.75)	16 (84.43)
aes (7)	1 (0.15)	**2 (76.52)**	**2 (72.25)**	**2 (73.34)**
atcoss (13)	5 (190.68)	5 (183.91)	5 (189.58)	5 (169.64)
bcp (16)	**16 (0.18)**	**16 (14.72)**	**16 (14.64)**	15 (3.39)
close_solutions (16)	16 (34.13)	16 (180.0)	16 (181.74)	16 (169.64)
decision-tree (13)	0 (-)	0 (-)	0 (-)	0 (-)
des (16)	**16 (458.86)**	14 (806.43)	15 (685.66)	13 (625.36)
drmx-atmost (16)	16 (20.61)	16 (197.65)	16 (197.65)	16 (263.4)
drmx-cryptogen (16)	0 (-)	0 (-)	0 (-)	**8 (664.19)**
exploits-synthesis_changjian_zhang (3)	2 (1231.87)	2 (1232.81)	2 (1170.36)	2 (1342.15)
extension-enforcement (16)	**8 (10.31)**	7 (163.06)	7 (175.49)	**8 (162.22)**
fault-diagnosis (16)	**16 (240.97)**	15 (86.09)	15 (85.16)	**16 (98.47)**
frb (16)	7 (4.4)	7 (118.28)	7 (117.49)	**8 (141.9)**
gen-hyper-tw (16)	3 (195.45)	3 (76.85)	3 (72.91)	3 (73.74)
HaplotypeAssembly (6)	**6 (22.99)**	5 (28.8)	**6 (196.76)**	**6 (86.75)**
hs-timetabling (1)	0 (-)	0 (-)	0 (-)	0 (-)
kbtree (14)	1 (0.04)	1 (1.52)	1 (1.5)	1 (1.54)
large-graph-community-detection_jabbour (8)	5 (0.01)	5 (0.38)	5 (0.52)	5 (0.49)
logic-synthesis (16)	10 (25.35)	**15 (22.19)**	**15 (22.16)**	**15 (22.07)**
maxclique (16)	13 (19.43)	13 (168.81)	13 (168.48)	13 (174.49)
maxcut (14)	0 (-)	0 (-)	0 (-)	0 (-)
MaximumCommonSub-GraphExtraction (16)	14 (182.42)	**16 (469.91)**	**16 (480.71)**	**16 (399.79)**
MaxSATQueriesInterpretableClassifiers (16)	9 (0.18)	9 (41.61)	9 (41.8)	**10 (297.56)**
mbd (16)	16 (3.55)	16 (166.18)	16 (165.3)	16 (166.33)
min-fill (16)	5 (152.92)	**6 (67.01)**	5 (25.22)	**6 (124.56)**
optic (16)	4 (1.77)	5 (714.54)	**6 (1065.25)**	**6 (412.63)**
phylogenetic-trees_berg (15)	14 (657.21)	14 (611.5)	14 (637.0)	14 (560.61)
planning-bnn (15)	**12 (1537.81)**	9 (1058.52)	10 (1119.94)	10 (1101.88)
program_disambiguation-Ramos (10)	8 (268.76)	8 (282.46)	8 (276.75)	8 (291.31)
protein_ins (12)	12 (123.26)	12 (255.34)	12 (257.83)	12 (227.07)
railroad_reisch (11)	0 (-)	0 (-)	0 (-)	0 (-)
railway-transport (6)	2 (9.73)	2 (58.25)	2 (54.89)	2 (51.85)
ramsey (2)	0 (-)	0 (-)	0 (-)	0 (-)
RBAC_marco.mori (16)	16 (12.86)	16 (90.17)	16 (92.85)	16 (77.84)
scheduling (5)	2 (1267.19)	2 (1120.45)	2 (1395.98)	2 (1263.36)
scheduling_xiaojuan (16)	5 (405.98)	**6 (788.28)**	5 (359.15)	**6 (654.7)**
SeanSafarpour (16)	**14 (171.39)**	12 (121.1)	12 (121.95)	12 (119.32)
security-witness_paxian (16)	16 (454.99)	16 (629.42)	16 (646.73)	16 (486.73)
set-covering (16)	0 (-)	0 (-)	0 (-)	0 (-)
setcover-rail_zhendong (4)	0 (-)	0 (-)	0 (-)	0 (-)
treewidth-computation (16)	13 (215.97)	**14 (369.28)**	**14 (335.56)**	**14 (347.69)**
uaq (16)	13 (219.37)	13 (467.97)	13 (454.21)	13 (318.79)
uaq_gazzarata (11)	**9 (235.22)**	8 (22.92)	8 (22.89)	8 (22.92)
xai-mindset2 (17)	11 (131.96)	11 (193.37)	11 (192.92)	11 (187.81)
reversi (5)	2 (0.25)	2 (6.52)	2 (6.37)	2 (6.13)
spinglass (2)	1 (7.36)	1 (182.23)	1 (182.32)	1 (183.63)
causal (16)	16 (133.39)	16 (146.93)	16 (148.32)	16 (143.08)

Table 6.7: Number of instances solved per benchmark family by the different techniques with IHS using non-optimal hitting sets (average solving time in seconds between brackets)

6.6. Portfolio Approach

The main question, based on the analysis of the previous section, becomes whether it is possible to create a portfolio strategy which determines which technique to use based on the (estimated) features of the instance. It should be noted that it is probable that the resulting portfolio approach has limited generalisability due to the limited number of instances. To determine which strategy to use, a decision tree is created to allow for easy interpretation while retaining the possibility of a non-linear decision boundary.

6.6.1. Feature Extraction

To be able to use supervised learning, a vector of features is required to be extracted. To this end, features are created based on certain characteristics of the instances, namely:

1. Average core length
2. Number of variables
3. Number of clauses
4. Total number of literals
5. Number of hard clauses
6. Number of soft clauses

These features are readily available for every file instance and can easily be extracted (with the exception of the Average core length). For the average core length, the average size of the cores found during the Core Generation Phase is used as an approximation. The label of each instance is based on which techniques solve a specific instance.

6.6.2. Decision Tree Construction

For creating the decision tree, the implementation makes use of the decision trees as implemented by the library scikit-learn [42]. To determine the parameters related to the criterion (gini, entropy or logarithmic loss), maximum tree depth (between 2 and 48), maximum number of leaf nodes (between 2 and 48) and the complexity parameter used for Minimal Cost-Complexity Pruning (ccp_alpha between 0 and 0.5), a grid search was used. This grid search shows that gini for the criterion, a maximum depth of 6, a maximum number of leaf nodes of 12 and a complexity parameter of 0 is optimal within the range of provided parameters. For IHS with non-optimal hitting sets the grid search shows that gini for the criterion, a maximum depth of 8, a maximum number of leaf nodes of 10 and a complexity parameter of 0 is optimal within the given range. For the cross-validation strategy, 5-fold cross-validation was used.

The resulting decision tree for "regular" IHS can be seen in Figure 6.8 (where # means "Number of") where M stands for Mixture_{CR+NI}, C stands for $\text{Community}_{NI+10CT}$ and S stands for Subset_{MO}. In the case where multiple techniques are present in a leaf node, the one with the most solved individual instances assigned to that leaf node is used. If all three techniques are present in a leaf node then the implementation defaults to the Subset selection technique due to its overall satisfactory performance. The decision tree for the techniques using IHS with non-optimal hitting sets can be seen in Figure 6.9.

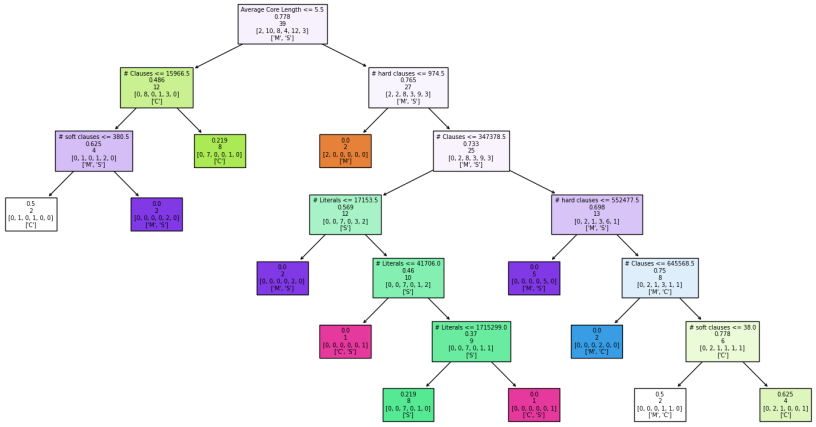


Figure 6.8: Diagram of decision tree for the Portfolio strategy

Judging by the fact that the root node in Figure 6.8 and Figure 6.9 is related to the average core length, this factor is influential in determining whether a specific instance is solved by a technique. As for the other metrics utilised in Figure 6.8, the diagram shows that the others are oftentimes influential in splitting the inner nodes further while the total number of literals is not present in any of the nodes as the deciding feature (though due to its inherent stochastic nature, a different random state might lead to a different decision tree in which this particular feature is used more frequently). Intuitively, the features used to represent an instance are oftentimes related to one another leading to a complex decision boundary. Interestingly, when observing Figure 6.9, it can be seen that the decision tree is similar in terms of its structure. However, the average core length seems to be present more often in Figure 6.9 as a deciding feature than in its "regular" counterpart indicating that for IHS with non-optimal hitting sets, the average core length is important for determining how an instance is classified.

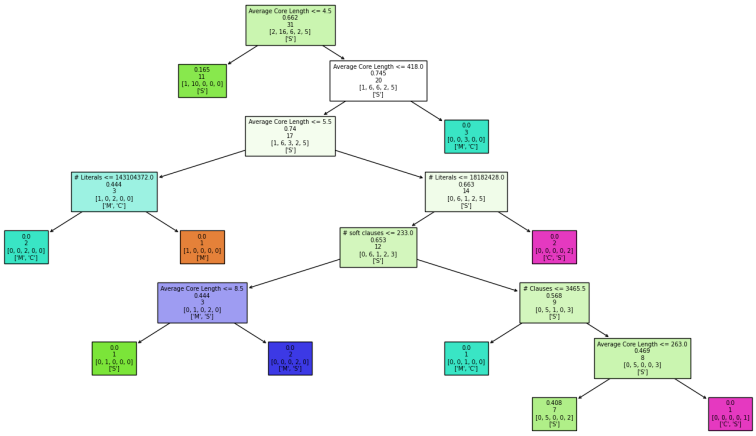


Figure 6.9: Diagram of decision tree for the Portfolio strategy using IHS with non-optimal hitting sets

6.6.3. Portfolio Results

The results of the portfolio strategy can be seen in Figure 6.10. In this case, Portfolio refers to the decision tree described in Figure 6.8 while Portfolio$_{NO}$ refers to the decision tree described in Figure 6.9.

While slightly outperforming `RC2`, the Portfolio approach can be seen to perform the worst out of both decision trees, solving a total of 375 instances. However, it should be noted that it does outperform `RC2` by a small margin. An explanation for this discrepancy is further explored in the following section.

Moreover, it can be observed that Portfolio$_{NO}$ performs the best out of both approaches, solving a total of 377 instances. Interestingly, both approaches fail to reach the performance of the best technique presented in Section 6.3 indicating that both oftentimes misidentify the appropriate technique for a given instance. This could indicate that the feature space which has been constructed does not appropriately capture the complexity of the MaxSAT instances. Adding additional information (such as graph metrics or engineered features) could potentially improve the performance of the Portfolio approach.

This discrepancy between the portfolio approaches further becomes apparent when analysing VBS$_{Portfolio}$ which solves a total of 387 instances. This indicates that there is either a significant amount of instances for which one of the strategies correctly identifies the instance but the other does not or there is a notable difference between which instances are solved by the different forms of IHS. As postulated previously, these misidentifications can potentially be avoided by introducing extra information to the feature space.

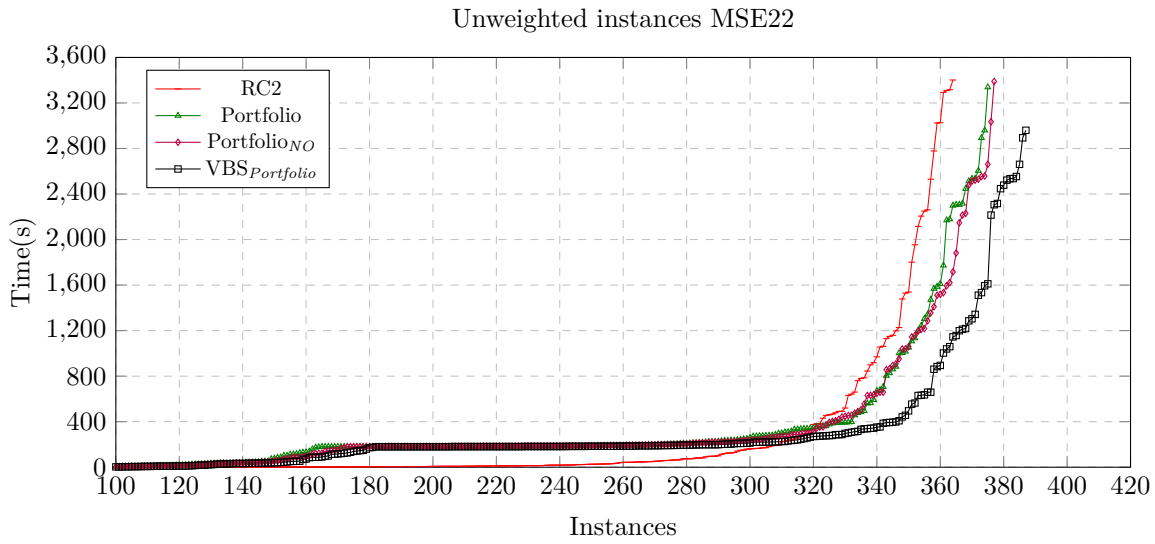


Figure 6.10: Cactus plot of Portfolio approach for MSE22 instances

6.6.4. Average Core Length Estimation

One of the main factors influencing the performance of the portfolio strategy is the accuracy of the estimate of the average core length (in combination with the accuracy of the decision tree itself). For both of the decision trees presented in the previous section(s), the average core length is oftentimes present in inner nodes which are encountered early during classification. This indicates that the average core length is likely crucial for determining the label assigned to an instance.

An overview of the metrics related to the error for the average core length estimation per IHS method can be seen in Table 6.8. The metrics are solely based on the instances which were not solved during the Core Generation Phase which could lead to different instances being considered for the different IHS methods. Nevertheless, it can be seen that the mean difference is quite significant. Both approaches oftentimes do not accurately estimate the core length due to a variety of reasons:

- The core minimization is inhibited by the conflict limit. Due to the nature of MaxSAT

solving, it is oftentimes undesirable to spend copious amounts of time minimizing cores while neglecting to solve the problem at hand. This leads to a conflict limit being imposed on the SAT-solver during the core minimization process. However, this conflict limit could result in cores which are not minimal which consequently causes an overestimation of the average core length.
- Furthermore, it could be that the cores which are oftentimes found during the initial call to the SAT-solver contain more variables than the cores found later in the search process. Due to the inherent time limit of the Core Generation Phase, this could then lead to an overestimation of the average core length.
- Finally, certain instances do not find a single core during the Core Generation Phase in which case the average core length estimation is equal to zero. This issue is increasingly apparent for instances for which the average core length is high and the first call to a SAT-solver is difficult to solve. It is likely that this is what causes both approaches to have the same maximum value.

The mean of the error indicates that the overestimation of the average core length is commonplace. The accuracy of the decision tree using these estimates can be seen in Table 6.8. While achieving accuracies of approximately 90% during training, it can be seen that both methods fail to achieve an accuracy of above 35% when using the estimates of the average core length. This indicates that the cores found during the Core Generation Phase are not necessarily representative of all the cores of the instance. This issue is likely to be remedied by a longer running time of the Core Generation Phase as the estimation of the average core length will become more accurate with more data.

IHS Method	Mean	Median	Min	Max	Accuracy Decision Tree
Regular	-549.76	0	-226859	10880	23%
Non-optimal	-590.17	0	-226859	10880	35%

Table 6.8: Metrics Error Average Core Length Estimation (Average Core Length - Estimate)

6.7. Chapter Conclusion

In conclusion, the performance of the hybrid algorithm is shown to be competitive with the performance of `RC2`, exceeding its performance on certain instances. The overall best-performing technique is Subset Selection with the Minimum Overlap selection strategy utilising IHS with non-optimal hitting sets. When using ”regular” IHS, the most effective combinations of techniques are as follows:

- **Mixture** - This technique performs best in combination with core rejection, non-incremental SAT solving and a Core Generation Phase of 720 seconds.
- **Community** - This technique (consisting of the Mixture technique with hypergraph splitting and core rejection) performs best in combination with non-incremental SAT solving and a hypergraph splitting threshold of 10%.
- **Subset** - This technique performs the best when making use of the Minimum Overlap (MO) strategy for selecting cores.

While both the Mixture technique and the Subset Selection technique marginally exceed the number of instances solved by `RC2`, this is not the case for the Community technique which solves slightly fewer instances than its counterparts.

Furthermore, experimentation indicates that all techniques except Mixture benefit from the usage of IHS with non-optimal hitting sets in comparison to ”regular” IHS. The technique which boasts the most significant increase in performance is the Subset Selection technique with the Minimum Overlap (increasing from 372 to 385). Interestingly, when comparing the translated lower bound with the actual lower bound found by IHS for this technique, the results show that the proximity of these two lower bounds is not necessarily the pivotal factor determining performance but rather that *which* cores are translated have a pronounced influence on the efficacy.

Moreover, analysing the instances which are solved solely by a single of the aforementioned techniques indicates that there are certain structures present which provide an indication of the efficacy of the different techniques. The Subset Selection technique appears to benefit from structures which either consist of numerous smaller cores or which consist of sizeable cores likely due to the fact that small cores have a decreased probability to overlap and the other techniques perform subpar on some of the instances with larger cores. The Community algorithm benefits most from structures which consist of smaller cores as this facilitates the effectiveness of the hypergraph splitting. Finally, the Mixture technique appears to represent a type of middling approach which prospers specifically on an instance for which neither of the other techniques performs well. The same conclusions can be drawn when making use of IHS with non-optimal hitting sets, however, the superlative performance of Subset Selection renders an analysis of the structures benefiting the other techniques obsolete.

An analysis of the performance of the techniques across different benchmark families further strengthens the hypotheses posed in the prior paragraph related to the applicability of the techniques. There are thus certain families of instances on which certain techniques excel compared to the other techniques. When analysing the performance of the techniques using IHS with non-optimal hitting sets, it becomes apparent that the discrepancies when compared to "regular" IHS are predominantly caused by the differences between the generated lower bounds. In certain cases, the amount of information which is discarded is significantly increased when using IHS with non-optimal hitting sets leading to subpar performance while in other cases the increased translated lower bound leads to a less strenuous Final Solving Phase.

Finally, a portfolio strategy based on decision trees is proposed and evaluated based on features of the graph such as the number of clauses and variables in addition to the size of the cores of the instance. While showing adequate performance in terms of the number of instances solved (outperforming `RC2`), it is nevertheless surpassed by the Subset Selection technique utilising IHS with non-optimal hitting sets. Whereas it achieves satisfactory accuracy during the training of the decision tree, in practice the inaccuracy of the estimation of the average core length causes irremediable hindrances in determining which technique to select thus causing subpar performance.

7

Conclusion

In this work, the issues inherent to simulating/translating between OLL and Core-guided search have been discussed.

Firstly, it has been shown that there are issues which arise due to the inherent disconnect between how OLL-solvers and IHS-solvers explore the solution space which impedes direct simulation. These issues arise from the inability to predict which variables are required to be present in the final solution solely based on an incomplete set of cores of the original formula. However, under the assumption that the possible minimum hitting sets discovered by an IHS-solver follow the same structure as the possible cardinality constraints created by an OLL-solver, it is possible to alleviate the discrepancies between the manner in which the potential solutions are generated by OLL and IHS. This algorithm works by creating cores such that the solutions to cardinality constraints correspond with minimum hitting sets. Under this assumption, the proposed algorithm is proven to be able to find the optimal solution given a set of translated cores.

These theoretical insights are followed by the description of a hybrid algorithm based on translating from the cores found by an IHS-solver to cores usable by an OLL-solver. For the translation strategy, two main techniques were proposed which consist of either selecting a set of disjoint cores or creating an OLL-core based on the connected components in the hypergraph using the information provided by the minimum hitting set. Furthermore, certain adjustments were proposed such as rejecting cores and clustering the hypergraph to address the shortcomings of the translated cores. These adjustments have been shown to practically improve performance on numerous instances. The evaluation shows that particular combinations of techniques provide similar or improved performance over the OLL-solver `RC2`. Moreover, IHS with non-optimal hitting sets for the Core Generation Phase was evaluated to be beneficial to the overall performance of the hybrid algorithm.

Finally, based on the performance of the different techniques, the instances were analysed with respect to metrics related to the core structure of the graph. This analysis was performed to discover any structures which indicate that a particular technique has improved performance compared to the others. It shows that, overall, there is some variation between the applicability of the techniques based on the number of cores generated and the size of the cores which have been generated. Utilising these results, a decision tree has been created to facilitate a portfolio strategy for deciding when to apply a certain technique. These results show that while the portfolio strategy slightly outperforms `RC2` and certain other strategies, there are certain factors that prevent it from performing better than other techniques such as the inaccuracy of the estimation of the features of an instance.

In conclusion, there is thus potential for a Core Generation Phase to improve the performance of an OLL-solver in terms of the number of instances solved, depending on the features of an instance. While there are many factors that influence the performance of such an algorithm, non-incremental SAT-solving in combination with IHS with non-optimal hitting sets is beneficial for the majority of techniques in terms of the number of instances solved while the effect of a prolonged Core Generation Phase is dependent on which technique is used.

8

Future work

Future work could consider several directions related to the work presented in this thesis:

1. Firstly, there are a significant number of hyperparameters which influence the performance of the hybrid algorithm. These include the duration of the Core Generation Phase, the thresholds considered for core rejection and hypergraph splitting, the IHS version which is used ("regular" or non-optimal), whether realizable hitting sets are created and the incrementality of the SAT-solver. This work presents the results related to a limited number of interactions between these components but future work could consider an extensive grid search to allow more granular analysis of the suitability of the different techniques.
2. Another direction could be related to the OLL-solver utilised in the Final Solving Phase. This work makes use of `RC2` due to its ease of implementation as opposed to other solvers which are (potentially) not as documented or which do not allow easy extensibility. However, the solver which is used has a significant impact on how well the techniques perform as different solvers oftentimes implement different strategies for core minimization and encodings of cardinality constraints. Future work could consider solvers such as `UWrMaxSAT`[9] and `PMRes` [7]. A similar argument holds for the underlying SAT-solver used by the OLL-solver as different SAT-solvers create different cores and a different number of cores which could lead to a significant influence on the results of the solver. Future work could make use of solvers such as `CaDiCal` [26].
3. Another consideration is the implementation of the IHS algorithm utilised during the Core Generation Phase. This work relies on an implementation based on the original MaxHS paper [10]. While performance appears adequate for the purposes of exhibiting the potential of the translation, an implementation in C++ or the addition of optimization techniques to the current implementation could potentially lead to significant performance gains.
4. Moreover, in this work, solely the unweighted instances are considered and `RC2` has been adjusted in such a way that it does not convert to a weighted instance (by assuring that if there exists a relaxation variable which has been seen before then it adds a fresh selector variable rather than raising the weight of the existing variable). A future research direction could consider whether the same performance benefits are present if weighted instances are considered. It is to be determined whether the techniques introduced in this work are also feasible directly in a weighted setting.
5. Finally, while this work proposes two techniques for translating the cores found by an IHS-solver to OLL-cores, these techniques are not all-encompassing and others could be considered. The techniques proposed in this work could serve as guidelines or baselines for the creation of other techniques. For example, a hybrid between subset selection and the hypergraph structure exploitation could be considered which utilises the strengths of both techniques.

References

[1] L. Zhang and F. Bacchus, "Maxsat heuristics for cost optimal planning", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, pp. 1846–1852, Sep. 2021. DOI: 10.1609/aaai.v26i1.8373. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/8373.

[2] J. Berg and M. Järvisalo, "Optimal correlation clustering via maxsat", in *2013 IEEE 13th International Conference on Data Mining Workshops*, 2013, pp. 750–757. DOI: 10.1109/ICDMW.2013.99.

[3] D. Malioutov and K. S. Meel, "Mlic: A maxsat-based framework for learning interpretable classification rules", in *Principles and Practice of Constraint Programming*, J. Hooker, Ed., Cham: Springer International Publishing, 2018, pp. 312–327, ISBN: 978-3-319-98334-9.

[4] J. Guerra and I. Lynce, "Reasoning over biological networks using maximum satisfiability", in *Principles and Practice of Constraint Programming*, M. Milano, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 941–956, ISBN: 978-3-642-33558-7.

[5] Z. Fu and S. Malik, "On solving the partial max-sat problem", in *Theory and Applications of Satisfiability Testing - SAT 2006*, A. Biere and C. P. Gomes, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 252–265, ISBN: 978-3-540-37207-3.

[6] A. Ignatiev, A. Morgado, and J. Marques-Silva, "Rc2: An efficient maxsat solver", *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, pp. 53–64, 2019, 1, ISSN: 1574-0617. DOI: 10.3233/SAT190116. [Online]. Available: https://doi.org/10.3233/SAT190116.

[7] N. Narodytska and F. Bacchus, "Maximum satisfiability using core-guided maxsat resolution", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, Jun. 2014. DOI: 10.1609/aaai.v28i1.9124. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/9124.

[8] A. Morgado, A. Ignatiev, and J. Marques-Silva, "Mscg: Robust core-guided maxsat solving", *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, pp. 129–134, 2014, 1, ISSN: 1574-0617. DOI: 10.3233/SAT190105. [Online]. Available: https://doi.org/10.3233/SAT190105.

[9] M. Piotrów, "Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems", in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2020, pp. 132–136. DOI: 10.1109/ICTAI50040.2020.00031.

[10] J. Davies, "Solving maxsat by decoupling optimization and satisfaction", Ph.D. dissertation, University of Toronto, 2013.

[11] J. Davies and F. Bacchus, "Postponing optimization to speed up maxsat solving", in *Principles and Practice of Constraint Programming*, C. Schulte, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 247–262, ISBN: 978-3-642-40627-0.

[12] J. Berg, F. Bacchus, and A. Poole, "Abstract cores in implicit hitting set maxsat solving (extended abstract)", in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed., Sister Conferences Best Papers, International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 4745–4749. DOI: 10.24963/ijcai.2021/643. [Online]. Available: https://doi.org/10.24963/ijcai.2021/643.

[13] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko, "Reduced cost fixing in maxsat", in *Principles and Practice of Constraint Programming*, J. C. Beck, Ed., Cham: Springer International Publishing, 2017, pp. 641–651, ISBN: 978-3-319-66158-2.

[14] D. Le Berre and A. Parrain, "The sat4j library, release 2.2", *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, pp. 59–64, 2010, 2-3, ISSN: 1574-0617. DOI: 10.3233/SAT190075. [Online]. Available: https://doi.org/10.3233/SAT190075.

[15] F. Heras, J. Larrosa, and A. Oliveras, "Minimaxsat: An efficient weighted max-sat solver", *Journal of Artificial Intelligence Research*, vol. 31, pp. 1–32, 2008.

[16] H. Lin, K. Su, and C.-M. Li, "Within-problem learning for efficient lower bound computation in max-sat solving", in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, ser. AAAI'08, Chicago, Illinois: AAAI Press, 2008, pp. 351–356, ISBN: 9781577353683.

[17] N. Narodytska and N. Bjørner, "Analysis of Core-Guided MaxSat Using Cores and Correction Sets", in *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, K. S. Meel and O. Strichman, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 236, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 26:1–26:20, ISBN: 978-3-95977-242-6. DOI: 10.4230/LIPIcs.SAT.2022.26. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2022/16700.

[18] B. Andres, B. Kaufmann, O. Matheis, and T. Schaub, "Unsatisfiability-based optimization in clasp", in *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[19] A. Gupta, M. K. Ganai, and C. Wang, "Sat-based verification methods and applications in hardware verification", in *Formal Methods for Hardware Verification: 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006, Bertinoro, Italy, May 22-27, 2006, Advanced Lectures 6*, Springer, 2006, pp. 108–143.

[20] M. D. Ernst, T. D. Millstein, and D. S. Weld, "Automatic sat-compilation of planning problems", in *IJCAI*, vol. 97, 1997, pp. 1169–1176.

[21] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving", *Formal methods in system design*, vol. 19, pp. 7–34, 2001.

[22] G. Audemard and L. Simon, "On the glucose sat solver", *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1 840 001, 2018.

[23] N. Eén and N. Sörensson, "Temporal induction by incremental sat solving", *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 4, pp. 543–560, 2003, BMC'2003, First International Workshop on Bounded Model Checking, ISSN: 1571-0661. DOI: https://doi.org/10.1016/S1571-0661(05)82542-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1571066105825423.

[24] S. Joshi, R. Martins, and V. Manquinho, "Generalized totalizer encoding for pseudo-boolean constraints", in *Lecture Notes in Computer Science*, Springer International Publishing, 2015, pp. 200–209. DOI: 10.1007/978-3-319-23219-5_15. [Online]. Available: https://doi.org/10.1007%2F978-3-319-23219-5_15.

[25] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "Qmaxsat: A partial max-sat solver", *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, pp. 95–100, 2012, 1-2, ISSN: 1574-0617. DOI: 10.3233/SAT190091. [Online]. Available: https://doi.org/10.3233/SAT190091.

[26] A. B. K. F. M. Fleury and M. Heisinger, "Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020", *SAT COMPETITION*, vol. 50, p. 2020, 2020.

[27] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization", *SAT*, vol. 2005, no. 53, pp. 1–2, 2005.

[28] C. Ansótegui, M. L. Bonet, and J. Levy, "Solving (weighted) partial maxsat through satisfiability testing", in *Theory and Applications of Satisfiability Testing - SAT 2009*, O. Kullmann, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 427–440, ISBN: 978-3-642-02777-2.

[29] J. Berg and M. Järvisalo, "Weight-aware core extraction in sat-based maxsat solving", in *International Conference on Principles and Practice of Constraint Programming*, Springer, 2017, pp. 652–670.

[30] J. Marques-Silva, "Minimal unsatisfiability: Models, algorithms and applications (invited paper)", in *2010 40th IEEE International Symposium on Multiple-Valued Logic*, 2010, pp. 9–14. DOI: 10.1109/ISMVL.2010.11.

[31] F. Bacchus and N. Narodytska, "Cores in core based maxsat algorithms: An analysis", in *Theory and Applications of Satisfiability Testing – SAT 2014*, C. Sinz and U. Egly, Eds., Cham: Springer International Publishing, 2014, pp. 7–15, ISBN: 978-3-319-09284-3.

[32] J. Berg, E. Demirović, and P. J. Stuckey, "Core-boosted linear search for incomplete maxsat", in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, L.-M. Rousseau and K. Stergiou, Eds., Cham: Springer International Publishing, 2019, pp. 39–56, ISBN: 978-3-030-19212-9.

[33] A. Ignatiev, A. Previti, M. Liffiton, and J. Marques-Silva, "Smallest mus extraction with minimal hitting set dualization", in *Principles and Practice of Constraint Programming*, G. Pesant, Ed., Cham: Springer International Publishing, 2015, pp. 173–182, ISBN: 978-3-319-23219-5.

[34] M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva, "Fast, flexible mus enumeration", *Constraints*, vol. 21, no. 2, pp. 223–250, Apr. 2016, ISSN: 1572-9354. DOI: 10.1007/s10601-015-9183-0. [Online]. Available: https://doi.org/10.1007/s10601-015-9183-0.

[35] B. S. Khan and M. A. Niazi, *Network community detection: A review and visual survey*, 2017. DOI: 10.48550/ARXIV.1708.00977. [Online]. Available: https://arxiv.org/abs/1708.00977.

[36] Y. Takai, A. Miyauchi, M. Ikeda, and Y. Yoshida, "Hypergraph clustering based on pagerank", in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20, Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 1970–1978, ISBN: 9781450379984. DOI: 10.1145/3394486.3403248. [Online]. Available: https://doi.org/10.1145/3394486.3403248.

[37] Y. Yang, S. Deng, J. Lu, *et al.*, "Graphlshc: Towards large scale spectral hypergraph clustering", *Information Sciences*, vol. 544, pp. 117–134, 2021, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2020.07.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520306824.

[38] T. Kumar, S. Vaidyanathan, H. Ananthapadmanabhan, S. Parthasarathy, and B. Ravindran, "A new measure of modularity in hypergraphs: Theoretical insights and implications for effective clustering", in *Complex Networks and Their Applications VIII*, H. Cherifi, S. Gaito, J. F. Mendes, E. Moro, and L. M. Rocha, Eds., Cham: Springer International Publishing, 2020, pp. 286–297, ISBN: 978-3-030-36687-2.

[39] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2023. [Online]. Available: https://www.gurobi.com.

[40] D. H. P. C. C. (DHPC), *DelftBlue Supercomputer (Phase 1)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[41] J. M. Hernández and P. Van Mieghem, "Classification of graph metrics", *Delft University of Technology: Mekelweg, The Netherlands*, pp. 1–20, 2011.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

A

Instance Families

```
{
        "aes-key-recovery": [
        "AES1-30-1.wcnf.xz",
        "AES1-30-2.wcnf.xz",
        "AES1-30-5.wcnf.xz",
        "AES1-50-15.wcnf.xz",
        "AES1-60-18.wcnf.xz",
        "AES1-60-19.wcnf.xz",
        "AES1-60-20.wcnf.xz",
        "AES1-70-21.wcnf.xz",
        "AES1-70-23.wcnf.xz",
        "AES1-70-25.wcnf.xz",
        "AES1-72-26.wcnf.xz",
        "AES1-74-34.wcnf.xz",
        "AES1-76-38.wcnf.xz",
        "AES2-40-50.wcnf.xz",
        "AES2-70-64.wcnf.xz",
        "AES2-74-75.wcnf.xz"
    ],
        "aes": [
        "mul_8_11.wcnf.xz",
        "mul_8_13.wcnf.xz",
        "mul_8_14.wcnf.xz",
        "mul_8_3.wcnf.xz",
        "mul_8_9.wcnf.xz",
        "sbox_4.wcnf.xz",
        "sbox_8.wcnf.xz"
    ],
    "atcoss": [
        "mes.atcoss_mesat_01.wcnf.xz",
        "mes.atcoss_mesat_04.wcnf.xz",
        "mes.atcoss_mesat_05.wcnf.xz",
        "mes.atcoss_mesat_06.wcnf.xz",
        "mes.atcoss_mesat_10.wcnf.xz",
        "sug.atcoss_sugar_01.wcnf.xz",
        "sug.atcoss_sugar_04.wcnf.xz",
        "sug.atcoss_sugar_05.wcnf.xz",
        "sug.atcoss_sugar_10.wcnf.xz",
        "sug.atcoss_sugar_11.wcnf.xz",
        "sug.atcoss_sugar_12.wcnf.xz",
        "sug.atcoss_sugar_15.wcnf.xz",
        "sug.atcoss_sugar_18.wcnf.xz"
    ],
    "bcp": [
        "hip-sim.simp-abcd_5.01.wcnf.xz",
        "hip-sim.simp-ace_25.06.wcnf.xz",
        "hip-sim.simp-cf_10.09.wcnf.xz",
        "hip-sim.simp-ibd_25.01.wcnf.xz",
        "hip-sim.simp-nonunif-100_100.11.wcnf.xz",
```

```
        "hip-sim.simp-nonunif-100_50.14.wcnf.xz",
        "hip-sim.simp-test_chr21_CEU_75.wcnf.xz",
        "hip-su.SU1__simp-genos.haps.48.wcnf.xz",
        "hip-su.SU3__simp-genos.haps.41.wcnf.xz",
        "mtg.apex2_Fv1@1.wcnf.xz",
        "mtg.c1355_F543gat@1.wcnf.xz",
                "msp.normalized-aim-100-1_6-yes1-2.wcnf.xz",
        "msp.normalized-aim-200-6_0-yes1-4.wcnf.xz",
        "msp.normalized-ii8b1.wcnf.xz",
        "msp.normalized-par16-3-c.wcnf.xz",
        "msp.normalized-ssa7552-038.wcnf.xz"
    ],
    "close_solutions": [
            "SAT02__industrial__goldberg__fpga_routing__vda_gr_rcs_w9.shuffled.cnf.wcnf
                .2.wcnf.xz",
        "SAT04__industrial__vangelder__cnf-color__abb313GPIA-9-tr.used-as.sat04-321.cnf.
            wcnf.6.wcnf.xz",
        "SAT04__industrial__velev__pipe-sat-1-1__12pipe_bug4_q0.used-as.sat04-723.cnf.
            wcnf.8.wcnf.xz",
        "SAT04__industrial__velev__pipe-sat-1-1__12pipe_bug6_q0.used-as.sat04-725.cnf.
            wcnf.7.wcnf.xz",
        "SAT09__APPLICATIONS__satComp09_BioInstances__rbcl_xits_15_SAT.cnf.wcnf.9.wcnf.
            xz",
        "SAT09__APPLICATIONS__satComp09_BioInstances__rbcl_xits_18_SAT.cnf.wcnf.2.wcnf.
            xz",
        "SAT11__application__fuhs__AProVE11__AProVE11-12.cnf.wcnf.5.wcnf.xz",
        "SAT11__application__fuhs__AProVE11__AProVE11-16.cnf.wcnf.1.wcnf.xz",
        "SAT11__application__jarvisalo__AAAI2010-SATPlanning__aaai10-planning-ipc5-
            pipesworld-12-step16.cnf.wcnf.2.wcnf.xz",
        "SAT11__application__jarvisalo__smtqfbv-aigs__smtlib-qfbv-aigs-rfunit_flat-64-
            tseitin.cnf.wcnf.3.wcnf.xz",
        "SAT11__application__leberre__2dimensionalstrippacking__E05F18.cnf.wcnf.9.wcnf.
            xz",
        "SAT11__application__manthey__traffic__traffic_r_sat.cnf.wcnf.4.wcnf.xz",
        "SAT11__application__manthey__traffic__traffic_r_sat.cnf.wcnf.6.wcnf.xz",
        "SAT11__application__rintanen__SATPlanning__blocks-blocks-36-0.180-SAT.cnf.wcnf
            .2.wcnf.xz",
        "SAT_RACE06__velev-pipe-sat-1.0-b9.cnf.wcnf.2.wcnf.xz",
                "teams16_l6a.cnf.wcnf.xz"
    ],
        "decision-tree": [
        "ann.formula_0.8_2021_atleast_31_max-4_reduced_incomplete_tree.wcnf.xz",
        "aus.formula_0.8_2021_atleast_31_max-4_reduced_incomplete_tree.wcnf.xz",
        "bre-res-str.formula_0.8_2021_atleast_127_max-6_reduced_incomplete_tree.wcnf.xz
            ",
        "car.formula_0.8_2021_atleast_127_max-6_reduced_incomplete_tree.wcnf.xz",
        "car.formula_0.8_2021_atleast_15_max-3_reduced_incomplete_tree.wcnf.xz",
        "hea.formula_0.8_2021_atleast_31_max-4_reduced_incomplete_tree.wcnf.xz",
        "hyp.formula_0.8_2021_atleast_31_max-4_reduced_incomplete_tree.wcnf.xz",
        "lym.formula_0.8_2021_atleast_31_max-4_reduced_incomplete_tree.wcnf.xz",
        "mus.formula_0.8_2021_atleast_15_max-3_reduced_incomplete_tree.wcnf.xz",
        "pri.formula_0.8_2021_atleast_63_max-5_reduced_incomplete_tree.wcnf.xz",
        "soy.formula_0.8_2021_atleast_127_max-6_reduced_incomplete_tree.wcnf.xz",
        "spl.formula_0.8_2021_atleast_63_max-5_reduced_incomplete_tree.wcnf.xz",
        "tic.formula_0.8_2021_atleast_15_max-3_reduced_incomplete_tree.wcnf.xz"
    ],
    "des": [
        "cnf.10.p.6.wcnf.xz",
        "cnf.10.p.7.wcnf.xz",
        "cnf.10.p.8.wcnf.xz",
        "cnf.13.p.6.wcnf.xz",
        "cnf.13.p.7.wcnf.xz",
        "cnf.14.p.4.wcnf.xz",
        "cnf.14.t.7.wcnf.xz",
        "cnf.15.p.6.wcnf.xz",
        "cnf.16.p.8.wcnf.xz",
        "cnf.17.p.8.wcnf.xz",
        "cnf.17.t.8.wcnf.xz",
        "cnf.18.p.10.wcnf.xz",
        "cnf.18.p.7.wcnf.xz",
```

```
        "cnf.19.p.5.wcnf.xz",
        "cnf.20.p.7.wcnf.xz",
        "cnf.20.p.9.wcnf.xz"
    ],
        "drmx-atmost": [
        "drmx-am12-outof-40-ekmtot.wcnf.xz",
        "drmx-am12-outof-40-emtot.wcnf.xz",
        "drmx-am16-outof-45-ecardn.wcnf.xz",
        "drmx-am16-outof-45-esortn.wcnf.xz",
        "drmx-am16-outof-45-etot.wcnf.xz",
        "drmx-am20-outof-50-emtot.wcnf.xz",
        "drmx-am20-outof-50-eseqc.wcnf.xz",
        "drmx-am20-outof-50-etot.wcnf.xz",
        "drmx-am24-outof-55-ecardn.wcnf.xz",
        "drmx-am24-outof-55-ekmtot.wcnf.xz",
        "drmx-am28-outof-60-ecardn.wcnf.xz",
        "drmx-am28-outof-60-emtot.wcnf.xz",
        "drmx-am32-outof-70-ekmtot.wcnf.xz",
        "drmx-am32-outof-70-eseqc.wcnf.xz",
        "drmx-am32-outof-70-esortn.wcnf.xz",
        "drmx-am32-outof-70-etot.wcnf.xz"
    ],
        "drmx-cryptogen": [
                "geffe128_1.wcnf.xz",
        "geffe128_6.wcnf.xz",
        "geffe128_9.wcnf.xz",
                "threshold128_2.wcnf.xz",
        "threshold128_3.wcnf.xz",
        "threshold128_4.wcnf.xz",
        "threshold128_5.wcnf.xz",
        "threshold128_8.wcnf.xz",
        "wolfram72_0.wcnf.xz",
        "wolfram72_1.wcnf.xz",
        "wolfram72_3.wcnf.xz",
        "wolfram72_5.wcnf.xz",
        "wolfram72_8.wcnf.xz",
        "wolfram72_9.wcnf.xz",
        "wolfram80_3.wcnf.xz",
        "wolfram80_5.wcnf.xz"
    ],
        "exploits-synthesis_changjian_zhang": [
                "flush_reload_min.wcnf.xz",
        "meltdown_min.wcnf.xz",
                "spectre_min.wcnf.xz"
    ],
    "extension-enforcement": [
        "extension-enforcement_non-strict_stb_150_0.1_2_8_2.wcnf.xz",
        "extension-enforcement_non-strict_stb_200_0.05_1_10_0.wcnf.xz",
        "extension-enforcement_non-strict_stb_200_0.05_1_10_1.wcnf.xz",
        "extension-enforcement_non-strict_stb_200_0.05_3_10_0.wcnf.xz",
        "extension-enforcement_non-strict_stb_200_0.05_3_10_4.wcnf.xz",
        "extension-enforcement_non-strict_stb_200_0.05_4_10_4.wcnf.xz",
        "extension-enforcement_non-strict_stb_200_0.1_2_10_1.wcnf.xz",
        "extension-enforcement_non-strict_stb_200_0.1_4_10_2.wcnf.xz",
        "extension-enforcement_strict_com_100_0.05_4_20_3.wcnf.xz",
        "extension-enforcement_strict_com_150_0.05_0_15_2.wcnf.xz",
        "extension-enforcement_strict_com_150_0.05_3_30_0.wcnf.xz",
        "extension-enforcement_strict_com_150_0.05_4_30_4.wcnf.xz",
        "extension-enforcement_strict_com_150_0.1_2_15_0.wcnf.xz",
        "extension-enforcement_strict_com_150_0.1_4_15_1.wcnf.xz",
        "extension-enforcement_strict_com_200_0.05_0_40_1.wcnf.xz",
        "extension-enforcement_strict_com_200_0.1_0_10_1.wcnf.xz"
    ],
        "fault-diagnosis": [
        "s01423_nan_explicit_3_0.wcnf.xz",
        "s01423_nan_explicit_9_0.wcnf.xz",
        "s38417_nan_explicit_10_0.wcnf.xz",
        "s38417_nan_explicit_13_0.wcnf.xz",
        "s38417_nan_explicit_14_0.wcnf.xz",
        "s38417_nan_explicit_18_0.wcnf.xz",
```

```
        "s38417_nan_explicit_1_0.wcnf.xz",
        "s38417_nan_explicit_29_0.wcnf.xz",
        "s38417_nan_explicit_8_0.wcnf.xz",
        "s38584_nan_explicit_12_0.wcnf.xz",
        "s38584_nan_explicit_16_0.wcnf.xz",
        "s38584_nan_explicit_17_0.wcnf.xz",
        "s38584_nan_explicit_24_0.wcnf.xz",
        "s38584_nan_explicit_39_0.wcnf.xz",
        "s38584_nan_explicit_44_0.wcnf.xz",
        "s38584_nan_explicit_7_0.wcnf.xz"
    ],
    "frb": [
                "frb20-11-2.partial.wcnf.xz",
        "frb20-11-3.partial.wcnf.xz",
        "frb20-11-4.partial.wcnf.xz",
        "frb20-11-5.partial.wcnf.xz",
        "frb25-13-2.partial.wcnf.xz",
        "frb25-13-5.partial.wcnf.xz",
        "frb30-15-1.partial.wcnf.xz",
        "frb30-15-2.partial.wcnf.xz",
        "frb30-15-5.partial.wcnf.xz",
        "frb35-17-2.partial.wcnf.xz",
        "frb35-17-3.partial.wcnf.xz",
        "frb40-19-3.partial.wcnf.xz",
        "frb40-19-5.partial.wcnf.xz",
                "frb10-6-2.wcnf.xz",
        "frb10-6-3.wcnf.xz",
        "frb10-6-5.wcnf.xz"
    ],
        "gen-hyper-tw": [
        "GenHyperTW_adder_15.wcnf.xz",
        "GenHyperTW_aim-50-1_6-no-3.wcnf.xz",
        "GenHyperTW_aim-50-1_6-yes1-3.wcnf.xz",
        "GenHyperTW_aim-50-3_4-yes1-3.wcnf.xz",
        "GenHyperTW_b01.wcnf.xz",
        "GenHyperTW_b06.wcnf.xz",
        "GenHyperTW_flat30-50.wcnf.xz",
        "GenHyperTW_flat30-99.wcnf.xz",
        "GenHyperTW_hole6.wcnf.xz",
        "GenHyperTW_hole8.wcnf.xz",
        "GenHyperTW_hole9.wcnf.xz",
        "GenHyperTW_par8-4-c.wcnf.xz",
        "GenHyperTW_par8-5-c.wcnf.xz",
        "GenHyperTW_s208.wcnf.xz",
        "GenHyperTW_s27.wcnf.xz",
        "GenHyperTW_uf20-050.wcnf.xz"
    ],
    "HaplotypeAssembly": [
        "splitedReads_0.matrix.wcnf.xz",
        "splitedReads_137.matrix.wcnf.xz",
        "splitedReads_158.matrix.wcnf.xz",
        "splitedReads_160.matrix.wcnf.xz",
        "splitedReads_18.matrix.wcnf.xz",
        "splitedReads_414.matrix.wcnf.xz"
    ],
        "hs-timetabling": [
                "GreeceWesternGreeceUniversityInstance4.xml.wcnf.xz"
    ],
        "kbtree": [
        "kbtree9_7_3_5_20_5.wcsp.wcnf.xz",
        "kbtree9_7_3_5_40_1.wcsp.wcnf.xz",
        "kbtree9_7_3_5_50_1.wcsp.wcnf.xz",
        "kbtree9_7_3_5_50_3.wcsp.wcnf.xz",
        "kbtree9_7_3_5_50_4.wcsp.wcnf.xz",
        "kbtree9_7_3_5_60_3.wcsp.wcnf.xz",
        "kbtree9_7_3_5_60_4.wcsp.wcnf.xz",
        "kbtree9_7_3_5_60_6.wcsp.wcnf.xz",
        "kbtree9_7_3_5_70_5.wcsp.wcnf.xz",
        "kbtree9_7_3_5_70_6.wcsp.wcnf.xz",
        "kbtree9_7_3_5_80_2.wcsp.wcnf.xz",
```

```
        "kbtree9_7_3_5_90_3.wcsp.wcnf.xz",
        "kbtree9_7_3_5_90_4.wcsp.wcnf.xz",
        "kbtree9_7_3_5_90_5.wcsp.wcnf.xz"
    ],
        "large-graph-community-detection_jabbour": [
        "AMAZON.wcnf.xz",
        "DBLP.wcnf.xz",
        "FOOTBALL.wcnf.xz",
        "KARATE.wcnf.xz",
        "POLITICSBOOK.wcnf.xz",
        "RAILWAY.wcnf.xz",
        "RISKMAP.wcnf.xz",
        "YOUTUBE.wcnf.xz"
    ],
        "logic-synthesis": [
        "normalized-5xp1.b.opb.msat.wcnf.xz",
        "normalized-9sym.b.opb.msat.wcnf.xz",
        "normalized-apex4.a.opb.msat.wcnf.xz",
        "normalized-bench1.pi.opb.msat.wcnf.xz",
        "normalized-clip.b.opb.msat.wcnf.xz",
        "normalized-count.b.opb.msat.wcnf.xz",
        "normalized-e64.b.opb.msat.wcnf.xz",
        "normalized-ex5.pi.opb.msat.wcnf.xz",
        "normalized-exam.pi.opb.msat.wcnf.xz",
        "normalized-f51m.b.opb.msat.wcnf.xz",
        "normalized-jac3.opb.msat.wcnf.xz",
        "normalized-max1024.pi.opb.msat.wcnf.xz",
        "normalized-prom2.pi.opb.msat.wcnf.xz",
        "normalized-rot.b.opb.msat.wcnf.xz",
        "normalized-sao2.b.opb.msat.wcnf.xz",
        "normalized-test4.pi.opb.msat.wcnf.xz"
    ],
        "maxclique": [
        "str.hamming8-2.clq.wcnf.xz",
        "str.johnson32-2-4.clq.wcnf.xz",
        "str.johnson8-2-4.clq.wcnf.xz",
        "str.p_hat1000-1.clq.wcnf.xz",
        "str.p_hat700-3.clq.wcnf.xz",
        "str.san200_0.9_1.clq.wcnf.xz",
                "ran.max_clq_150-13-6258-1.clq.wcnf.xz",
        "ran.max_clq_150-14-6705-2.clq.wcnf.xz",
        "ran.max_clq_150-16-7599-3.clq.wcnf.xz",
        "ran.max_clq_150-17-8046-3.clq.wcnf.xz",
        "ran.max_clq_150-3-1788-2.clq.wcnf.xz",
        "ran.max_clq_150-3-1788-3.clq.wcnf.xz",
        "ran.max_clq_150-4-2235-1.clq.wcnf.xz",
        "ran.max_clq_150-7-3576-2.clq.wcnf.xz",
        "ran.max_clq_150-8-4023-3.clq.wcnf.xz",
        "ran.max_clq_150-9-4470-1.clq.wcnf.xz"
    ],
        "maxcut": [
        "bip.maxcut-140-630-0.7-13.wcnf.xz",
        "bip.maxcut-140-630-0.7-37.wcnf.xz",
        "bip.maxcut-140-630-0.7-42.wcnf.xz",
        "bip.maxcut-140-630-0.7-44.wcnf.xz",
        "bip.maxcut-140-630-0.8-24.wcnf.xz",
                "ran.max_cut_60_500_10.asc.wcnf.xz",
        "ran.max_cut_60_500_6.asc.wcnf.xz",
        "ran.max_cut_60_600_4.asc.wcnf.xz",
        "ran.max_cut_60_600_9.asc.wcnf.xz",
                "dim.brock400_3.clq.wcnf.xz",
        "dim.brock400_4.clq.wcnf.xz",
        "dim.hamming10-4.clq.wcnf.xz",
        "dim.hamming8-2.clq.wcnf.xz",
                "dim.MANN_a9.clq.wcnf.xz"
    ],
    "MaximumCommonSub-GraphExtraction": [
        "sol.g2_n26e25_n34e40.wcnf.xz",
        "sol.g2_n32e31_n35e41.wcnf.xz",
        "sol.g2_n43e42_n90e107.wcnf.xz",
```

```
        "sol.g2_n60e83_n77e84.wcnf.xz",
        "sol.g2_n74e91_n76e86.wcnf.xz",
        "sol.g3_n25e24_n35e34_n37e37.wcnf.xz",
        "sol.g3_n30e43_n33e37_n40e53.wcnf.xz",
        "sol.g3_n37e47_n38e49_n54e63.wcnf.xz",
                "sol.g4_n28e40_n47e56_n47e56_n58e77.wcnf.xz",
        "sol.g4_n34e43_n59e60_n69e85_n288e287.wcnf.xz",
        "sol.g4_n42e55_n45e54_n55e66_n78e101.wcnf.xz",
        "sol.g5_n32e44_n33e41_n35e48_n48e62_n50e64.wcnf.xz",
        "sol.g5_n39e38_n43e43_n61e61_n67e68_n82e119.wcnf.xz",
        "sol.g5_n41e56_n49e61_n51e60_n99e132_n109e132.wcnf.xz",
        "sol.g5_n67e69_n67e69_n82e119_n92e101_n92e101.wcnf.xz",
        "sol.g7_n35e48_n39e54_n41e57_n42e58_n47e66_n53e75_n59e84.wcnf.xz"
    ],
        "MaxSATQueriesInterpretableClassifiers": [
        "MLI.ionosphere_train_1_DNF_4_1.wcnf.xz",
        "MLI.twitter_train_1_CNF_4_1.wcnf.xz",
        "wcn.adult_test_9_CNF_1_1.wcnf.xz",
        "wcn.compas_test_3_DNF_5_1.wcnf.xz",
        "wcn.heart_test_9_CNF_4_1.wcnf.xz",
        "wcn.heart_train_6_DNF_3_1.wcnf.xz",
        "wcn.ionosphere_test_8_CNF_2_1.wcnf.xz",
        "wcn.ionosphere_train_8_CNF_4_1.wcnf.xz",
        "wcn.ionosphere_train_8_DNF_5_1.wcnf.xz",
        "wcn.parkinsons_train_3_DNF_2_1.wcnf.xz",
        "wcn.parkinsons_train_7_CNF_2_1.wcnf.xz",
        "wcn.pima_test_4_CNF_2_1.wcnf.xz",
        "wcn.tictactoe_test_9_DNF_4_1.wcnf.xz",
        "wcn.tictactoe_train_7_DNF_4_1.wcnf.xz",
        "wcn.toms_test_4_CNF_3_1.wcnf.xz",
        "wcn.transfusion_train_7_CNF_2_1.wcnf.xz"
    ],
    "mbd": [
        "b14_C-mbd14-0248.wcnf.xz",
        "b15_C-mbd14-0259.wcnf.xz",
        "b17_C-mbd14-0261.wcnf.xz",
        "b20_C-mbd14-0252.wcnf.xz",
        "b20_C-mbd14-0267.wcnf.xz",
        "b20_C-mbd14-0298.wcnf.xz",
        "b20_C-mbd14-0358.wcnf.xz",
        "b21_C-mbd14-0250.wcnf.xz",
        "b21_C-mbd14-0295.wcnf.xz",
        "b21_C-mbd14-0304.wcnf.xz",
        "b21_C-mbd14-0308.wcnf.xz",
        "b21_C-mbd14-0394.wcnf.xz",
        "b22_C-mbd14-0225.wcnf.xz",
        "b22_C-mbd14-0250.wcnf.xz",
        "b22_C-mbd14-0308.wcnf.xz",
        "b22_C-mbd14-0318.wcnf.xz"
    ],
        "min-fill": [
        "MinFill_R0_miles1500.wcnf.xz",
        "MinFill_R0_mulsol.i.3.wcnf.xz",
        "MinFill_R0_mulsol.i.4.wcnf.xz",
        "MinFill_R0_mulsol.i.5.wcnf.xz",
        "MinFill_R0_myciel4.wcnf.xz",
        "MinFill_R0_myciel5.wcnf.xz",
        "MinFill_R0_myciel7.wcnf.xz",
        "MinFill_R0_queen11_11.wcnf.xz",
        "MinFill_R0_queen6_6.wcnf.xz",
        "MinFill_R0_queen7_7.wcnf.xz",
        "MinFill_R10_anna.wcnf.xz",
        "MinFill_R3_david.wcnf.xz",
        "MinFill_R3_huck.wcnf.xz",
        "MinFill_R3_miles750.wcnf.xz",
        "MinFill_R4_miles500.wcnf.xz",
        "MinFill_R5_jean.wcnf.xz"
    ],
        "optic": [
        "gen_cvc-add3-carry2-gadget_33.wcnf.xz",
```

```
                "gen_add_4_33.wcnf.xz",
        "gen_add_6_991.wcnf.xz",
        "gen_add_4_carry_33.wcnf.xz",
        "gen_add_4_carry_991.wcnf.xz",
        "gen_add_5_carry_299.wcnf.xz",
        "gen_cvc-add7to3_991.wcnf.xz",
        "gen_cvc-mult4_4_991.wcnf.xz",
        "gen_cvc-mult4_4_9999.wcnf.xz",
        "gen_mult_4_4_9999.wcnf.xz",
        "gen_mult_4_5_299.wcnf.xz",
        "gen_mult_4_5_9999.wcnf.xz",
        "gen_mult_4_6_991.wcnf.xz",
        "gen_mult_4_7_991.wcnf.xz",
        "gen_mult_5_5_399.wcnf.xz",
        "gen_mult_5_5_991.wcnf.xz"
    ],
    "phylogenetic-trees_berg": [
        "ms_100_20_20-0.wcnf.xz",
        "ms_100_40_10-0.wcnf.xz",
        "ms_150_10_40-0.wcnf.xz",
        "ms_200_10_40-0.wcnf.xz",
        "ms_200_20_12-1.wcnf.xz",
        "ms_200_20_20-8.wcnf.xz",
        "ms_200_20_24-1.wcnf.xz",
        "ms_200_20_24-2.wcnf.xz",
        "ms_200_22_20-0.wcnf.xz",
        "ms_200_22_20-3.wcnf.xz",
        "ms_200_36_20-1.wcnf.xz",
        "ms_200_40_10-0.wcnf.xz",
        "ms_200_6_20-0.wcnf.xz",
        "ms_250_20_20-0.wcnf.xz",
        "ms_300_20_20-0.wcnf.xz"
    ],
        "planning-bnn": [
        "cellda_x_10.wcnf.xz",
        "cellda_x_12.wcnf.xz",
        "cellda_y_10.wcnf.xz",
        "sysadmin_4_3.wcnf.xz",
        "sysadmin_4_4.wcnf.xz",
        "sysadmin_5_3.wcnf.xz",
        "sysadmin_5_4.wcnf.xz",
        "navigation_3x3_5.wcnf.xz",
        "navigation_3x3_6.wcnf.xz",
        "navigation_4x4_5.wcnf.xz",
        "navigation_4x4_6.wcnf.xz",
        "navigation_4x4_7.wcnf.xz",
        "navigation_5x5_10.wcnf.xz",
        "navigation_5x5_8.wcnf.xz",
        "navigation_5x5_9.wcnf.xz"
    ],
    "program_disambiguation-Ramos": [
        "Q12_OPTIONS_1.wcnf.xz",
        "Q13_OPTIONS_1.wcnf.xz",
        "Q34_YESNO_2.wcnf.xz",
        "Q41_YESNO_3.wcnf.xz",
        "Q45_YESNO_3.wcnf.xz",
        "Q46_OPTIONS_1.wcnf.xz",
        "Q46_YESNO_2.wcnf.xz",
        "Q47_YESNO_2.wcnf.xz",
        "Q53_YESNO_3.wcnf.xz",
        "Q6_YESNO_1.wcnf.xz"
    ],
    "protein_ins": [
        "1bpi_.2knt_.g.wcnf.t.wcnf.xz",
        "1bpi_.5pti_.g.wcnf.t.wcnf.xz",
        "1knt_.1bpi_.g.wcnf.t.wcnf.xz",
        "1knt_.2knt_.g.wcnf.t.wcnf.xz",
        "1knt_.5pti_.g.wcnf.t.wcnf.xz",
        "1vii_.1cph_.g.wcnf.t.wcnf.xz",
        "2knt_.5pti_.g.wcnf.t.wcnf.xz",
```

```
        "3ebx_.1era_.g.wcnf.t.wcnf.xz",
        "3ebx_.6ebx_.g.wcnf.t.wcnf.xz",
        "6ebx_.1era_.g.wcnf.t.wcnf.xz",
        "sandiaprotein.g.wcnf.t.wcnf.xz",
                "p1.wcnf.t.wcnf.xz"
    ],
        "railroad_reisch": [
        "unw.MultiDay_2.wcnf.xz",
        "unw.MultiDay_3.wcnf.xz",
        "unw.MultiDay_4.wcnf.xz",
        "unw.SingleDay_15.wcnf.xz",
        "unw.SingleDay_3.wcnf.xz",
        "unw.SingleDay_37.wcnf.xz",
        "unw.Subnetwork_7.wcnf.xz",
        "unw.Subnetwork_9.wcnf.xz",
        "wei.MultiDay_0_weighted.wcnf.xz",
        "wei.MultiDay_1_weighted.wcnf.xz",
        "wei.SingleDay_2_weighted.wcnf.xz"
    ],
        "railway-transport": [
        "d4.wcnf.xz",
                "dp43.wcnf.xz",
                "p15.wcnf.xz",
           "pesp_18Min.wcnf.xz",
        "pesp_5min.wcnf.xz",
                "we.wcnf.xz"
    ],
        "ramsey": [
        "ram_k3_n19.ra0.wcnf.xz",
        "ram_k4_n18.ra0.wcnf.xz"
    ],
        "RBAC_marco.mori": [
        "mul.role_domino_multiple_0.0_1.wcnf.xz",
        "mul.role_domino_multiple_0.0_4.wcnf.xz",
        "mul.role_smallcomp_multiple_0.0_2.wcnf.xz",
        "mul.role_smallcomp_multiple_0.0_3.wcnf.xz",
        "mul.role_smallcomp_multiple_0.0_4.wcnf.xz",
        "mul.role_university_multiple_0.0_2.wcnf.xz",
        "mul.role_university_multiple_0.0_8.wcnf.xz",
        "vio.role_domino_violations_0.0_12.wcnf.xz",
        "vio.role_domino_violations_0.0_3.wcnf.xz",
        "vio.role_domino_violations_0.0_9.wcnf.xz",
        "vio.role_smallcomp_violations_0.0_3.wcnf.xz",
        "vio.role_smallcomp_violations_0.0_4.wcnf.xz",
        "vio.role_smallcomp_violations_0.0_5.wcnf.xz",
        "vio.role_university_violations_0.0_0.wcnf.xz",
        "vio.role_university_violations_0.0_3.wcnf.xz",
        "vio.role_university_violations_0.0_5.wcnf.xz"
    ],
        "scheduling": [
        "cnf_10_center.wcnf.xz",
        "cnf_12_center.wcnf.xz",
        "cnf_small.wcnf.xz",
                "cnf_10.wcnf.xz",
        "cnf_12.wcnf.xz"
    ],
    "scheduling_xiaojuan": [
        "lam.20-100-frag12-0.wcnf.xz",
        "lam.20-100-frag12-45.wcnf.xz",
        "lam.20-100-frag12-56.wcnf.xz",
        "lam.20-100-frag12-79.wcnf.xz",
        "lam.20-100-lambda100-53.wcnf.xz",
        "lam.20-100-lambda100-59.wcnf.xz",
        "lam.20-100-lambda100-89.wcnf.xz",
        "lam.20-100-lambda100-95.wcnf.xz",
        "lam.20-100-p100-55.wcnf.xz",
        "lam.20-100-p100-72.wcnf.xz",
        "lam.20-100-p100-86.wcnf.xz",
        "lam.20-100-p100-97.wcnf.xz",
        "lam.20-500-18.wcnf.xz",
```

```
        "lam.20-500-55.wcnf.xz",
        "lam.20-500-74.wcnf.xz",
        "lam.20-500-89.wcnf.xz"
    ],
        "SeanSafarpour": [
        "divider-problem.dimacs_3.filtered.wcnf.xz",
        "fpu4-problem.dimacs_19.filtered.wcnf.xz",
        "fpu5-problem.dimacs_20.filtered.wcnf.xz",
        "i2c_master1.dimacs.filtered.wcnf.xz",
        "mem_ctrl1.dimacs.filtered.wcnf.xz",
        "rsdecoder-problem.dimacs_41.filtered.wcnf.xz",
        "rsdecoder1.dimacs.filtered.wcnf.xz",
        "rsdecoder2.dimacs.filtered.wcnf.xz",
        "wb_4m8s-problem.dimacs_47.filtered.wcnf.xz",
                "b14_opt_bug2_vec1-gate-0.dimacs.seq.filtered.wcnf.xz",
        "c1_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtered.wcnf.xz",
        "c2_DD_s3_f1_e2_v1-bug-onevec-gate-0.dimacs.seq.filtered.wcnf.xz",
        "c5_DD_s3_f1_e1_v2-bug-gate-0.dimacs.seq.filtered.wcnf.xz",
        "c7552-bug-gate-0.dimacs.seq.filtered.wcnf.xz",
        "rsdecoder1_blackbox_KESblock-problem.dimacs_30.filtered.wcnf.xz",
        "s15850-bug-onevec-gate-0.dimacs.seq.filtered.wcnf.xz"
    ],
        "security-witness_paxian": [
        "unw.RSN_Security_Min_Witness-Direct-MBIST_100cores_100controllers_5memories_na-
            D2.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-MBIST_100cores_100controllers_5memories_na-
            D3.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-MBIST_100cores_100controllers_5memories_na-
            D6.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-MBIST_100cores_100controllers_5memories_na-
            D7.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-MBIST_100cores_100controllers_5memories_na-
            DC.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-MBIST_100cores_100controllers_5memories_na-
            DD.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-MBIST_5cores_100controllers_100memories_na-
            D1.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-MBIST_5cores_100controllers_100memories_na-
            D7.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-MBIST_5cores_100controllers_100memories_na-
            DE.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-FlexScan-D2.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-FlexScan-D4.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-FlexScan-D6.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-FlexScan-D7.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-FlexScan-DC.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-FlexScan-DD.wcnf.xz",
        "unw.RSN_Security_Min_Witness-Direct-p93791-D7.wcnf.xz"
    ],
    "set-covering": [
        "cra-scp.scpclr13_maxsat.wcnf.xz",
        "cra-scp.scpcyc10_maxsat.wcnf.xz",
        "ran-scp.scp43_maxsat.wcnf.xz",
        "ran-scp.scp47_maxsat.wcnf.xz",
        "ran-scp.scp48_maxsat.wcnf.xz",
        "ran-scp.scp510_maxsat.wcnf.xz",
        "ran-scp.scp58_maxsat.wcnf.xz",
        "ran-scp.scp59_maxsat.wcnf.xz",
        "ran-scp.scp61_maxsat.wcnf.xz",
        "ran-scp.scp65_maxsat.wcnf.xz",
        "ran-scp.scpnre1_maxsat.wcnf.xz",
        "ran-scp.scpnrf4_maxsat.wcnf.xz",
        "ran-scp.scpnrf5_maxsat.wcnf.xz",
        "ran-scp.scpnrg1_maxsat.wcnf.xz",
        "ran-scp.scpnrh1_maxsat.wcnf.xz",
        "ran-scp.scpnrh5_maxsat.wcnf.xz"
    ],
        "setcover-rail_zhendong": [
                "data.135.wcnf.xz",
        "data.243.wcnf.xz",
```

```
        "data.405.wcnf.xz",
        "data.729.wcnf.xz"
    ],
    "treewidth-computation": [
        "TWComp_barley-pp_N26.wcnf.xz",
        "TWComp_barley2_N48.wcnf.xz",
        "TWComp_celar02_N100.wcnf.xz",
        "TWComp_celar09pp_N67.wcnf.xz",
        "TWComp_david-pp_N29.wcnf.xz",
        "TWComp_david_N87.wcnf.xz",
        "TWComp_hailfinder_N56.wcnf.xz",
        "TWComp_hepar2_N70.wcnf.xz",
        "TWComp_mildew_35.wcnf.xz",
        "TWComp_miles500_N128.wcnf.xz",
        "TWComp_mulsol.i.5-pp_N119.wcnf.xz",
        "TWComp_myciel4_N23.wcnf.xz",
        "TWComp_myciel5_N47.wcnf.xz",
        "TWComp_oesoca4_N42.wcnf.xz",
        "TWComp_queen5_5_N25.wcnf.xz",
        "TWComp_win95pts_N76.wcnf.xz"
    ],
    "uaq": [
        "uaq-nr-nr110-nc36-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-nr-nr160-nc53-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-nr-nr170-nc56-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-nr-nr180-nc60-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-nr-nr220-nc73-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-nr-nr240-nc80-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-nr-nr260-nc86-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-nr-nr360-nc120-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-nr-nr470-nc156-n3-k2-rpp4-ppr2-plb50.wcnf.xz",
        "uaq-plb-nr250-nc83-n5-k2-rpp3-ppr5-plb90.wcnf.xz",
        "uaq-ppr-nr200-nc66-n5-k2-rpp4-ppr14-plb100.wcnf.xz",
        "uaq-ppr-nr200-nc66-n5-k2-rpp4-ppr6-plb100.wcnf.xz",
        "uaq-ppr-nr200-nc66-n5-k2-rpp6-ppr10-plb100.wcnf.xz",
        "uaq-ppr-nr200-nc66-n5-k2-rpp6-ppr7-plb100.wcnf.xz",
        "uaq-rpp-nr100-nc33-n3-k2-rpp29-ppr2-plb20.wcnf.xz",
        "uaq-rpp-nr100-nc33-n3-k2-rpp30-ppr2-plb20.wcnf.xz"
    ],
        "uaq_gazzarata": [
        "max.uaq-max-nc-nr10-np400-rpp5-nc10-n8-t3-plb10-n0.wcnf.xz",
        "max.uaq-max-nc-nr10-np400-rpp5-nc100-n8-t3-plb10-n4.wcnf.xz",
        "max.uaq-max-np-nr200-np700-rpp5-nc50-n8-t3-plb10-n5.wcnf.xz",
        "max.uaq-max-nr-nr160-np400-rpp5-nc50-n8-t3-plb10-n7.wcnf.xz",
        "max.uaq-max-nr-nr70-np400-rpp5-nc50-n8-t3-plb10-n7.wcnf.xz",
        "min.uaq-min-nr-nr50-np400-rpp5-nc0-rs0-t0-plb2-n4.wcnf.xz",
        "min.uaq-min-nr-nr70-np400-rpp5-nc0-rs0-t0-plb2-n9.wcnf.xz",
        "min.uaq-min-plb-nr10-np400-rpp5-nc0-rs0-t0-plb40-n0.wcnf.xz",
        "min.uaq-min-rpp-nr200-np400-rpp2-nc0-rs0-t0-plb4-n3.wcnf.xz",
        "min.uaq-min-rpp-nr200-np400-rpp3-nc0-rs0-t0-plb4-n4.wcnf.xz",
        "min.uaq-min-rpp-nr200-np400-rpp4-nc0-rs0-t0-plb4-n4.wcnf.xz"
    ],
        "xai-mindset2": [
                "australian.wcnf.xz",
                "cleveland.wcnf.xz",
                "cloud.wcnf.xz",
                "glass.wcnf.xz",
                "iris.wcnf.xz",
                "liver-disorder.wcnf.xz",
                "new-thyroid.wcnf.xz",
                "dermatology.wcnf.xz",
        "promoters.wcnf.xz",
        "titanic.wcnf.xz",
                "uci_mammo_data.wcnf.xz",
        "bupa.wcnf.xz",
        "colic.wcnf.xz",
        "corral.wcnf.xz",
        "ecoli.wcnf.xz",
        "indo.wcnf.xz",
        "hepatitis.wcnf.xz"
```

```
    ],
        "reversi": [
        "rev44-10.wcnf.xz",
        "rev66-14.wcnf.xz",
        "rev66-16.wcnf.xz",
        "rev66-20.wcnf.xz",
        "rev66-4.wcnf.xz"
    ],
        "spinglass": [
        "spi-Han.spinglass4_9.pm3.wcnf.xz",
        "spi-Han.spinglass5_2.pm3.wcnf.xz"
    ],
        "causal": [
        "causal_n5_i5_N10000_uai13_constant_int.wcnf.xz",
        "causal_n5_i7_N1000_uai13_constant_int.wcnf.xz",
        "causal_n5_i9_N10000_uai13_harddeps_int.wcnf.xz",
        "causal_n5_i9_N1000_uai13_constant_int.wcnf.xz",
        "causal_n5_i9_N1000_uai13_harddeps_int.wcnf.xz",
        "causal_n6_i10_N500_uai14_constant_int.wcnf.xz",
        "causal_n6_i2_N10000_uai14_harddeps_int.wcnf.xz",
        "causal_n6_i2_N1000_uai13_constant_int.wcnf.xz",
        "causal_n6_i8_N10000_uai14_constant_int.wcnf.xz",
        "causal_n6_i8_N1000_uai14_constant_int.wcnf.xz",
        "causal_n7_i10_N1000_uai14_constant_int.wcnf.xz",
        "causal_n7_i5_N10000_uai14_constant_int.wcnf.xz",
        "causal_n7_i5_N1000_uai14_constant_int.wcnf.xz",
        "causal_n7_i6_N500_uai14_harddeps_int.wcnf.xz",
        "causal_n7_i8_N1000_uai14_constant_int.wcnf.xz",
        "causal_n7_i8_N1000_uai14_harddeps_int.wcnf.xz"
    ]
}
```