UpMax: User Partitioning for MaxSAT

Pedro Orvalho ✉ ⓘ
INESC-ID, Instituto Superior Técnico, University of Lisbon, Portugal

Vasco Manquinho ✉ ⓘ
INESC-ID, Instituto Superior Técnico, University of Lisbon, Portugal

Ruben Martins ✉ ⓘ
Carnegie Mellon University, Pittsburgh, PA, USA

────── **Abstract** ──────

It has been shown that Maximum Satisfiability (MaxSAT) problem instances can be effectively solved by partitioning the set of soft clauses into several disjoint sets. The partitioning methods can be based on clause weights (e.g., stratification) or based on graph representations of the formula. Afterwards, a merge procedure is applied to guarantee that an optimal solution is found.

This paper proposes a new framework called UPMAX that decouples the partitioning procedure from the MaxSAT solving algorithms. As a result, new partitioning procedures can be defined independently of the MaxSAT algorithm to be used. Moreover, this decoupling also allows users that build new MaxSAT formulas to propose partition schemes based on knowledge of the problem to be solved. We illustrate this approach using several problems and show that partitioning has a large impact on the performance of unsatisfiability-based MaxSAT algorithms.

1 Introduction

In the last decade, Maximum Satisfiability (MaxSAT) algorithmic improvements have resulted in the successful usage of MaxSAT algorithms in several application domains such as fault localization [23], scheduling [14], planning [55], data analysis [9], among other [22, 51, 18]. These improvements resulted from new algorithm designs [39] based on iterative calls to a highly efficient Satisfiability (SAT) solver. However, MaxSAT algorithms also take advantage of other techniques, such as effective encodings of cardinality constraints [53] or the incremental usage of SAT solvers [34].

Another technique for MaxSAT solving is to use partitioning on the soft clauses. For instance, several solvers use partitioning of soft clauses according to their weight [4], which are particularly effective when the MaxSAT instance encodes a lexicographic optimization problem [32]. For the particular case of partial MaxSAT, other techniques have been proposed, such as using a graph representation of the formula [42]. However, despite its success for some classes of benchmarks, graph-based partitioning has not been widely used mainly because (1) the graph representation may become too large to build or to process,
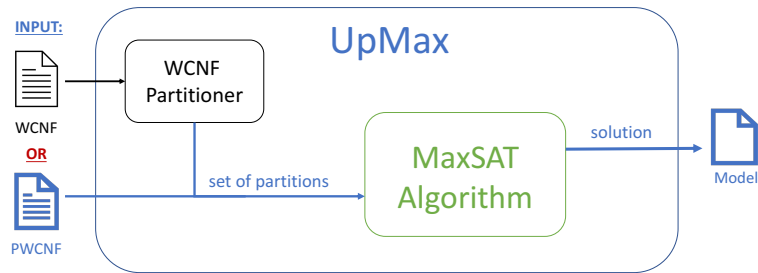
Figure 1 Overview of UpMax.

and (2) it is not decoupled with the base MaxSAT algorithm (i.e., changing the partition method implies altering the MaxSAT algorithm). Furthermore, in some cases, the partitions might not capture the problem structure that is helpful for MaxSAT solving. Since several MaxSAT algorithms rely on identifying unsatisfiable subformulas, each partition should be an approximation of an unsatisfiable subformula to be solved separately.

Until now, the partitioning of MaxSAT formulas is interconnected to the subsequent algorithm to be used. Therefore, it is not easy to define and test new partitioning methods with several MaxSAT algorithms developed by different people. The first contribution of this work is to propose a new format called `pwcnf` for defining MaxSAT formulas where clauses are split into partitions. Figure 1 illustrates the schematic view of the UpMax architecture based on the decoupling of the MaxSAT solving algorithm from the split of the clauses on the MaxSAT formula. Observe that any partitioning method (e.g., graph-based partitioning [42]) can be used to generate the instances in the new `pwcnf` format. Hence, this new format allows decoupling of the partitioning procedure from the MaxSAT algorithm, facilitating the appearance of new partition methods for MaxSAT formulas. Secondly, UpMax is not restricted to any partitioning scheme. UpMax also allows the MaxSAT user to propose how to partition MaxSAT formulas based on her domain knowledge of the problem to be solved. Note that this is not possible with current MaxSAT tools. Thirdly, with little effort, MaxSAT algorithms based on unsatisfiability approaches can be adapted to the new format. This is possible due to the newly proposed UpMax architecture. Hence, this paper presents the results of several algorithms using different partitioning schemes. Finally, we present several use cases where different user-defined partitioning schemes can be easily defined and tested. Experimental results show that user-based partitioning significantly impacts the performance of MaxSAT algorithms. Thus, UpMax decouples clause partitioning from MaxSAT solving, opening new research directions for partitioning, modeling, and algorithm development.

2   Background

A propositional formula in Conjunctive Normal Form (CNF) is defined as a conjunction of clauses where a clause is a disjunction of literals such that a literal is either a propositional variable v_i or its negation $\neg v_i$. Given a CNF formula ϕ, the Satisfiability (SAT) problem corresponds to decide if there is an assignment such that ϕ is satisfied or prove that no such assignment exists. The Maximum Satisfiability (MaxSAT) is an optimization version of the SAT problem. Given a CNF formula ϕ, the goal is to find an assignment that minimizes the number of unsatisfied clauses in ϕ. In partial MaxSAT, clauses in ϕ are split in hard ϕ_h and soft ϕ_s. Given a formula $\phi = (\phi_h, \phi_s)$, the goal is to find an assignment that satisfies all hard clauses in ϕ_h while minimizing the number of unsatisfied soft clauses in ϕ_s. The partial

■ **Algorithm 1** Generic Partition-based MaxSAT Algorithm.

Input: $\phi = (\phi_h, \phi_s)$
Output: optimal assignment to ϕ
1 $\gamma \leftarrow \langle \gamma_1, \ldots, \gamma_n \rangle \leftarrow \texttt{partitionSoft}(\phi_h, \phi_s)$ `// initial partitions`
2 **if** $|\gamma| = 1$ **then**
3 $\quad$ **return** MaxSAT (ϕ_h, ϕ_s) `// no partitions`
4 **foreach** $\gamma_i \in \gamma$ **do**
5 $\quad$ $\nu \leftarrow \texttt{MaxSAT}(\phi_h, \gamma_i)$
6 **while** true **do**
7 $\quad$ $(\gamma_i, \gamma_j) \leftarrow \texttt{selectPartitions}(\gamma)$
8 $\quad$ $\gamma_k \leftarrow \texttt{mergePartitions}(\gamma_i, \gamma_j)$
9 $\quad$ $\gamma \leftarrow \gamma \setminus \{\gamma_i, \gamma_j\} \cup \{\gamma_k\}$ `// update partition set`
10 $\quad$ $\nu \leftarrow \texttt{MaxSAT}(\phi_h, \gamma_k)$
11 $\quad$ **if** $|\gamma| = 1$ **then**
12 $\quad\quad$ **return** ν

MaxSAT problem can be further generalized to the weighted version, where each soft clause has an associated weight, and the optimization goal is to minimize the sum of the weights of the unsatisfied soft clauses. Finally, we assume that ϕ_h is satisfiable. Moreover, the set notation is also commonly used to manipulate formulas and clauses, i.e., a CNF formula can be seen as a set of clauses (its conjunction), and a clause as a set of literals (its disjunction).

2.1 Algorithms for MaxSAT

Currently, state of the art algorithms for MaxSAT are based on successive calls to a SAT solver [39]. One of the approaches is to perform a SAT-UNSAT linear search on the number of unsatisfied soft clauses. For that, the algorithm starts by adding a new relaxation variable r_i to each soft clause $s_i \in \phi_s$, where r_i represents the unsatisfiability of clause s_i. Next, it defines an initial upper bound μ on the number of unsatisfied soft clauses. At each SAT call, the constraint $\sum r_i \leq \mu - 1$ is added such that an assignment that improves on the previous one is found. Whenever the working formula becomes unsatisfiable, then the previous SAT call identified an optimal assignment. There are a plethora of these algorithms [26, 53, 44, 45].

On the other hand, UNSAT-SAT algorithms start with a lower bound λ on the number of unsatisfied soft clauses initialized at 0. The algorithm starts with an overconstrained working formula. At each iteration, the working formula is relaxed by adding additional relaxation variables allowing more soft clauses to be unsatisfied. Whenever the working formula becomes satisfiable, then an optimal solution is found. There are also many successful MaxSAT solvers that use an UNSAT-SAT approach [16, 4, 5, 31, 33, 40, 12]. Two key factors for the performance of UNSAT-SAT algorithms is the usage of SAT solvers to identify unsatisfiable subformulas, and the search process being incremental [15, 34]. Instead of dealing with the whole formula at once, some algorithms try to split the formula into partitions [4, 35, 42]. In particular, partitioning focuses on splitting the set of soft clauses into disjoint sets. The motivation is to quickly identify a minimal cost considering just a subset of soft clauses. Since the sets are disjoint, the sum of the minimal cost of all partitions defines a lower bound on the optimal solution. Moreover, a smaller instance should be able to be easier to solve. Hence, the convergence to the optimum is expected to be faster.

Algorithm 1 presents the pseudo-code for this generic approach. First, soft clauses in ϕ_s are split into n disjoint sets (line 1). If $n = 1$, then there is no partitioning and a MaxSAT solver is called on the whole formula. Otherwise, a MaxSAT solver solves each

Hard: $h_1 : (v_1 \lor v_2)$ $h_2 : (\neg v_2 \lor v_3)$ $h_3 : (\neg v_1 \lor \neg v_3)$ $h_4 : (v_4 \lor v_5)$

  $h_5 : (\neg v_5 \lor v_6)$ $h_6 : (\neg v_4 \lor \neg v_6)$ $h_7 : (\neg v_3 \lor \neg v_6)$

Soft: $s_1 : (\neg v_1)$ $s_2 : (\neg v_3)$ $s_3 : (\neg v_4)$ $s_4 : (\neg v_6)$

■ **Figure 2** Example of a MaxSAT formula.
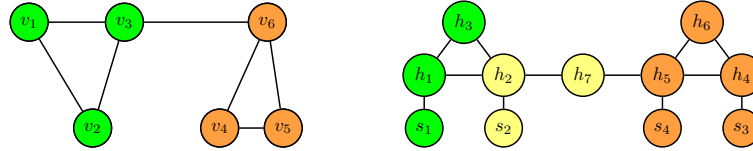
■ **Figure 3** VIG graph (left) and RES graph (right) for MaxSAT formula in Figure 2.

partition γ_i independently (line 5). Next, two partitions are selected and merged (lines 7-8). The newly merged partition γ_k is then solved considering the information already obtained from solving γ_i and γ_j. This process is repeated until there is only one partition whose solution is an optimal assignment to the original MaxSAT instance (line 12). Observe that several MaxSAT algorithms can be used in this scheme including Fu-Malik [16], WPM3 [6], MSU3 [33], OLL [40] or a hitting set approach [12, 47], among others [39].

▶ **Example 1.** Consider the MaxSAT formula in Figure 2. Suppose the soft clauses are split into two disjoint sets $\gamma_1 = \{s_1, s_2\}$ and $\gamma_2 = \{s_3, s_4\}$. Next, a MaxSAT solver is applied to MaxSAT instances (ϕ_h, γ_1) and (ϕ_h, γ_2). Each of these instances has an optimal solution of 1. When merging both partitions, a final MaxSAT call is made on $(\phi_h, \gamma_1 \cup \gamma_2)$ with an initial lower bound of 2 (because γ_1 and γ_2 are disjoint). Since the lower bound is already equal to the optimum value, this last call is not likely to be computationally hard.

A related approach to partitioning is *Group MaxSAT* [7, 17]. Group MaxSAT is a variation of MaxSAT where soft clauses are grouped, and each group has a weight. The optimization goal in Group MaxSAT is to minimize the sum of the weights of the unsatisfied groups. A group is considered unsatisfied if at least one of its soft clauses is unsatisfied. Note that Group MaxSAT and MaxSAT are solving different optimization problems. The partitions in Algorithm 1 do not change the optimization goal of MaxSAT but instead are meant to guide the solver to find an optimal solution to the MaxSAT formula.

2.2 Partitioning MaxSAT Formulas

There are several graph representations for CNF formulas that have been proposed in order to analyze its structural properties [52, 50, 3]. For instance, it is well-known that industrial SAT instances can be represented in graphs with high modularity [3]. On the other hand, graphs that represent randomly generated instances are closer to an Erdös-Rényi model. Similar observations have also been made in MaxSAT instances [42]. Furthermore, based on these graph representations, one can partition the set of soft clauses in a MaxSAT instance by applying a community finding algorithm [10] that maximizes the modularity value.

One possible graph representation is the Variable Incidence Graph (VIG). Let $G = (V, E)$ denote a weighted undirected graph where V defines the graph vertices and E its edges. In the VIG representation, we have a vertex $v_i \in V$ for each variable v_i in the MaxSAT formula ϕ. Next, for each pair of variables v_i and v_j, if there is at least one clause in ϕ that contains both variables v_i and v_j (or its negated literals), then an edge (v_i, v_j) is added to the graph. For each clause $c \in \phi$ with n literals, then $1/\binom{n}{2}$ is added to the weight of every

pair of variables that occurs in clause c. Figure 3 illustrates the VIG representation for the MaxSAT formula in Figure 2. Note that edge weights are not represented to simplify the figure. Next, if we apply a community finding algorithm that maximizes the modularity, two communities are identified (vertices with different colors). Hence, soft clauses with variables in $\{v_1, v_2, v_3\}$ would define a partition, while soft clauses with variables in $\{v_4, v_5, v_6\}$ define the other partition. Therefore, we would have $\gamma_1 = \{s_1, s_2\}$ and $\gamma_2 = \{s_3, s_4\}$ as the two partitions of soft clauses. Observe that $\phi_h \cup \gamma_1$ is an unsatisfiable subformula of ϕ, as well as $\phi_h \cup \gamma_2$. In the Clause-Variable Incidence Graph (CVIG) representation, there is a node for each variable and another node for each clause. Moreover, if a variable v_i (or its negation $\neg v_i$) occurs in a clause c_j, then there is an edge (v_i, c_j) in the graph.

On the other hand, in Resolution-based Graphs (RES) only clauses are represented as vertices. Hence, for each clause c_j there is a node in the graph. Let c_{jk}^r be the clause that results from applying the resolution operation between clauses c_j and c_k. If c_{jk}^r is not a tautology, then an edge (c_j, c_k) is added to the graph with weight $1/|c_{jk}^r|$ where $|c_{jk}^r|$ denotes the size of the resolvent clause. Note that if the resolution operation results in a trivial resolvent, then no edge is added. The right graph in Figure 3 shows the RES graph representation of the MaxSAT formula in Figure 2. Colors illustrate the three communities of soft clauses found in this formula, i.e., $\gamma_1 = \{s_1\}$, $\gamma_2 = \{s_2\}$ and $\gamma_3 = \{s_3, s_4\}$.

3 User-Based Partitioning

All the partitioning methods described in Section 2.2 are automatic and attempt to recover some partition scheme from the structure of the MaxSAT formula. However, when encoding a problem into MaxSAT, it is often the case that the user has enough domain knowledge to provide a potential partition scheme. Unfortunately, the current MaxSAT format does not support this extra information. Thus, we propose a new generic format for MaxSAT, `pwcnf`, where the partition scheme can be defined, and solvers can take advantage of it. The `pwcnf` format starts with a header:

`p pwcnf n_vars n_clauses topw n_part`

and each line in the body is of the form: `[part] [weight] [literals*] 0`.

In the header, `n_vars` and `n_clauses` are the numbers of variables and clauses of the formula. `topw` is the weight assigned to the hard clauses and `n_part` is the number of partitions. Each partition label (`[part]`) must be a positive integer from 1 to `n_part`.

▶ **Example 2.** Considering the RES graph presented in Figure 3, clearly it has 3 partitions accordingly to the coloring scheme: green (label 1), yellow (label 2) and orange (label 3). The `pwcnf` for this graph is the following:

```
p pwcnf 6 11 7 3        1 7 -1 -3 0    3 7 -4 -6 0    2 1 -3 0
1 7 1 2 0               3 7 4 5 0      2 7 -3 -6 0    3 1 -4 0
2 7 -2 3 0              3 7 -5 6 0     1 1 -1 0       3 1 -6 0
```

Alloy [21] is a declarative modeling language based on a first-order relational logic that has been applied to different software engineering problems [25, 30, 24, 49]. Recently, Alloy^{Max} has been proposed that extends Alloy with the ability to find optimal solutions [54]. Alloy^{Max} uses a MaxSAT solver as its optimization engine. So for that, Alloy^{Max} encodes in a MaxSAT formula the high-level Alloy model. Moreover, Alloy^{Max} uses domain knowledge to partition the soft clauses in the generated MaxSAT formula. In particular, one partition of soft clauses is created for each optimization operator used in the Alloy specification. Alloy^{Max} was the first successful usage of UPMAX, which shows that the creation of the new file format makes it easier for other researchers to integrate partitioning in their applications.

3.1   Use Case: Minimum Sum Coloring

To illustrate how a user can take advantage of the new proposed format, we analyze the Minimum Sum Coloring (MSC) problem from graph theory. MSC is the problem of finding a proper coloring while minimizing the sum of the colors assigned to the vertices. In this problem, the following conditions must be met: (1) each vertex should be assigned a color; (2) each vertex is assigned at most one color; and (3) two adjacent vertices cannot be assigned the same color. The optimization goal is to minimize the number of different colors in the graph. Let V denote the set of vertices in the graph. Let C denote the set of possible colors. Let X_c^v be the Boolean that is assigned to 1 if color c is assigned to vertex v. The goal is to maximize $\neg X_c^v$, and each soft clause is assigned the weight of c. The user can potentially group these soft clauses in two distinct ways: (1) variables that share the same color are grouped, or (2) variables that share the vertex number are grouped. For more details on the MaxSAT encoding, the reader is referred to the extended version of this paper [43].

▶ **Example 3.** Assume a user wants to minimize the number of different colors needed to color a given graph G such that two adjacent vertices cannot share the same color. G has 4 vertices, $v_1, \ldots, v_4$, and the following set of edges $G_E = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_3, v_4)\}$. Furthermore, there are 4 different colors available $c_1, \ldots, c_4$. When encoding the problem into `pwcnf` the user could provide either the following VERTEX-based or COLOR-based partition scheme:
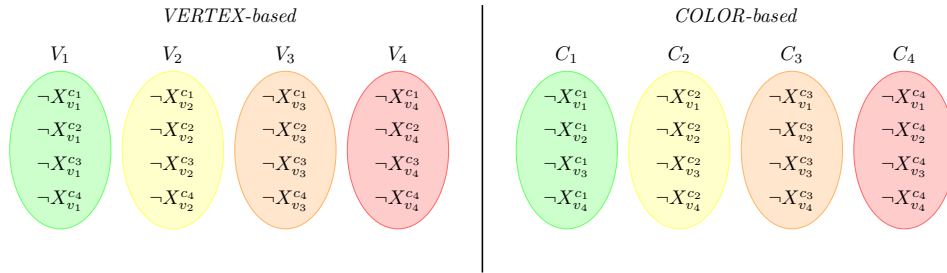
3.2   Use Case: Seating Assignment Problem

Another example of how a user can take advantage of the `pwcnf` format is the seating assignment problem. We encode this problem into MaxSAT and show different partition schemes that can be provided by the user. Consider a seating assignment problem where the goal is to seat persons at tables such that the following properties are met: (1) Each table has a minimum and a maximum number of persons; (2) Each person is seated at exactly one table; and (3) Each person has some tags that represent their interests. The optimization goal is to minimize the number of different tags between all persons seated at the same table.

More formally, consider a seating problem with p persons and t tables. Assume that the set of tags each person may have is defined by G and the set of tables is defined by T. Consider the Boolean variables Y_t^g that are assigned to 1 if there is *at least one person* p with a tag g that is seated at table t. The goal of this optimization problem is to minimize $\sum_{t \in T, g \in G} Y_t^g$ subject to the constraints of the problem. When encoding the problem into MaxSAT, the $\neg Y_t^g$ literals will correspond to unit soft clauses. The user can potentially group these soft clauses in two distinct ways: (1) variables that share the same tag are grouped, or (2) variables that share the same table are grouped. We call the former partition scheme *TAGS-based* and the latter *TABLES-based*. More details can be found in the extended version of this paper [43].

▶ **Example 4.** Consider that a user wants to seat 5 persons, $p_1, \ldots, p_5$, in two tables t_1, t_2. Each table must have at least 2 persons and at most 3 persons. Each person has a set of interests described by their tags as follows:

$$p_1 = \{A, B\}, p_2 = \{C\}, p_3 = \{B\}, p_4 = \{C, A\}, p_5 = \{A\}$$

When encoding the problem into `pwcnf` the user could provide either the following TAGS-based or TABLES-based partition scheme:

TAGS-based | *TABLES-based*

A B C t_1 t_2

$\neg Y_{t_1}^A$ $\neg Y_{t_1}^B$ $\neg Y_{t_1}^C$ $\neg Y_{t_1}^A$ $\neg Y_{t_2}^A$

$\neg Y_{t_2}^A$ $\neg Y_{t_2}^B$ $\neg Y_{t_2}^C$ $\neg Y_{t_1}^B$ $\neg Y_{t_2}^B$

$\neg Y_{t_1}^C$ $\neg Y_{t_2}^C$
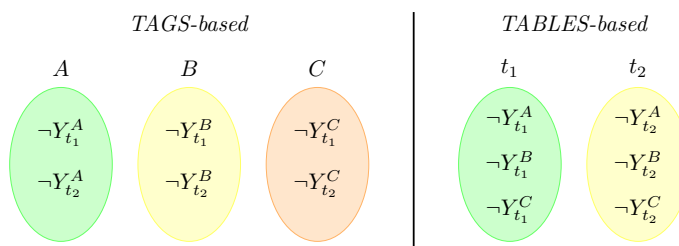
4   Experimental Results

UpMax is built on top of the open-source Open-WBO MaxSAT solver [36]. UpMax supports the new format `pwcnf` for user-based partitioning presented in Section 3. Alternatively, it can also take as input a `wcnf` formula and output a `pwcnf` formula using an automatic partitioning strategy based on VIG [37], CVIG [37], RES [42], or randomly splitting the formula into k partitions. UpMax can also be extended to support additional partitioning strategies that users may want to implement to evaluate their impact on the performance of MaxSAT algorithms. Furthermore, we are currently merging the partitions based on their size. We sort the partitions in increasing order, and we start by giving the smaller partition to the solver. However, other merging methods can be easily implemented and evaluated.

UpMax currently supports three UNSAT-based algorithms (WBO [31], OLL [40], and MSU3 [34]) for both unweighted and weighted problems that take advantage of the partitions using the basic algorithm described in Algorithm 1. WBO uses only at-most-one cardinality constraints when relaxing the formula at each iteration. In contrast, MSU3 uses a single cardinality constraint, and OLL uses multiple cardinality constraints. Furthermore, we have also extended RC2 [20] and Hitman [38], available in PySAT [19], to take advantage of user-based or graph-based partitions through our `pwcnf` formulae using Algorithm 1. RC2 is an improved version of the OLL algorithm [40, 41]. Hitman is a SAT-based implementation of an implicit minimal hitting set enumerator [38] and can be used as the basic flow of a MaxHS-like algorithm [11] for MaxSAT. UpMax is publicly available at GitHub [1].

To show the impact of partitioning on the performance of unsatisfiability-based algorithms, we randomly generate 1,000 instances for both the seating assignment and the minimum sum coloring problem by varying the different parameters of each problem. We considered the automatic partitioning strategies available in UpMax (VIG, CVIG, RES, random; for the random partitioning strategy, we fixed $k = 16$), the user partitions (UP) described in Sections 3.1 and 3.2, and no partitions. All of the experiments were run on StarExec [48] with a timeout of 1800 seconds and a memory limit of 32 GB for each instance.

4.1   Minimum Sum Coloring Problem

Table 1 shows the number of instances solved for the Minimum Sum Coloring problem for each partitioning strategy and algorithm. Entries highlighted in bold correspond to the highest number of instances solved for each algorithm. The diversity of algorithms and

■ **Table 1** Number of solved instances for the Minimum Sum Coloring (MSC) problem.

		User Part.		Graph Part.			
Solver	**No Part.**	**Vertex**	**Color**	**VIG**	**CVIG**	**RES**	**Random**
MSU3	245	758	770	774	770	775	**776**
OLL	796	863	594	945	944	**947**	756
WBO	483	622	314	745	750	**755**	493
Hitman	610	613	471	605	**614**	609	592
RC2	796	866	528	943	939	**944**	687

■ **Table 2** Number of solved instances for the Seating Assignment problem.

|		User Part.		Graph Part.			
Solver	**No Part.**	**Table**	**Tag**	**VIG**	**CVIG**	**RES**	**Random**
MSU3	558	**671**	639	659	641	640	565
OLL	526	**634**	624	627	599	608	528
WBO	306	400	**536**	400	385	386	360
Hitman	420	403	**510**	406	425	420	440
RC2	530	620	**624**	618	600	597	541

partitions used allows us to make some interesting observations regarding the impact of partitioning on the performance of MaxSAT algorithms. First, we can see that partitioning MaxSAT algorithms can often significantly outperform their non-partitioning counterparts. For instance, with partitioning the WBO algorithm can solve 272 more instances than without partitioning. Secondly, most partition schemes result in performance improvements, even if the partition is done randomly. In this problem, random partitions had a benefit for the MSU3 algorithm. This occurs since, until the last partition is added, this algorithm deals with a subset of soft clauses, resulting in finding smaller unsatisfiable cores. Another observation is that the user-based partitions were not as good as the graph-based partitions. This may be partially explained by the fact that the user-based partitions do not consider the weight of the soft clauses which is important for weighted MaxSAT algorithms. Finally, we can also observe that different partition strategies have different performance impacts on different algorithms. This suggests that new algorithms could leverage the partition information better than our approach presented in Algorithm 1.

Figures 4a and 4b show two scatter plots comparing two MaxSAT algorithms, OLL and WBO, on the set of instances for the Minimum Sum Coloring problem. Each point in Figure 4a represents an instance where the x-value (resp. y-value) is the CPU time spent to solve the instance using the OLL algorithm with the RES partitioning scheme (resp. OLL without any partitioning scheme). If a point is above the diagonal, then it means that the algorithm with partitioning outperformed the algorithm without partitioning. The OLL algorithm was the one with the best performance in the Minimum Sum Coloring (MSC) set of instances (see Table 1). For many instances, we can observe a $10\times$ speedup for OLL-RES when compared to OLL-NoPart. Secondly, Figure 4b compares the WBO algorithm with no partitioning and WBO using the RES partitioning scheme. WBO has the most significant gap between using partitions (755 instances solved) and not using partitions (483 instances solved). Figure 4b also shows that many instances that could not be solved without partitions can now be solved in a few seconds. Both plots support that OLL and WBO greatly improve their performance on this set of benchmarks when using partitioning. Cactus plots of our experiments can be found in the extended version of this paper [43].
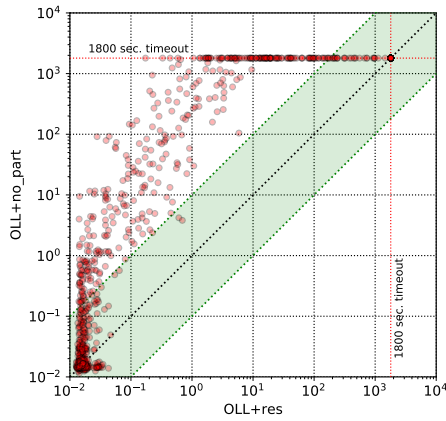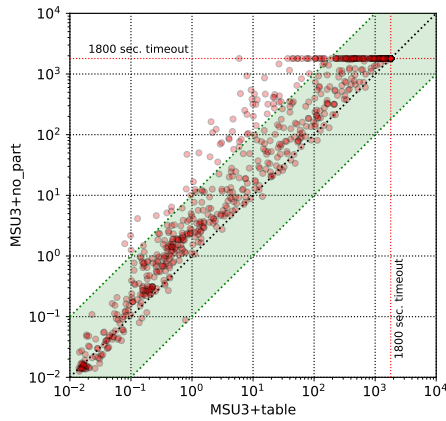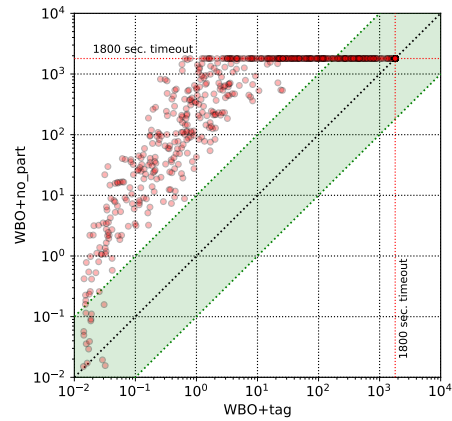
(a) MSC – OLL RES VS No Part.
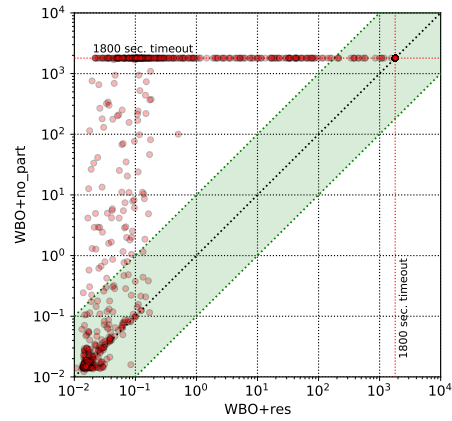
(b) MSC – WBO RES VS No Part.

(c) SA – MSU3 Table VS No Part.

(d) SA – WBO Tag VS No Part.

Figure 4 Scatter plots comparing different MaxSAT algorithms and respective partitioning schemes for the problems of Minimum Sum Coloring (MSC) and Seating Assignment (SA).

4.2 Seating Assignment Problem

Table 2 shows the number of instances solved for the Seating Assignment problem for each partitioning strategy and algorithm. Similar to the previous use case, we can observe that partitioning can significantly impact the number of solved instances for all evaluated MaxSAT algorithms. For instance, WBO algorithm with partitioning can solve more 220 instances than without partitioning. Furthermore, almost all the presented partition schemes, except random partitioning, result in performance improvements when compared to no partitioning scheme. Moreover, different partition strategies have different performance impacts on different algorithms. In this problem, user-based partitions achieved the best results. However, in some algorithms (e.g., MSU3, OLL), tabled-based partitioning is the best approach, while tag-based partitioning is better for the other algorithms.

Figures 4c and 4d show two scatter plots comparing two MaxSAT algorithms, MSU3 and WBO, on the set of instances for the Seating Assignment problem. Figure 4c shows the effectiveness of the table-based partitioning scheme against not using partitions in the MSU3 algorithm. We can observe that partitioning leads to faster runtimes, with most points being above the diagonal. Secondly, Figure 4d compares the WBO algorithm with the tag-based

and without any partitioning scheme. This algorithm has the biggest gap between using partitions (536 instances solved) and not using partitions (306 instances solved). Figure 4d supports that using partitioning with the WBO algorithm on this set of benchmarks greatly improves its performance with speedups of more than $10\times$ for most of the instances.

4.3   State-of-the-art MaxSAT Solvers

Using the `wcnf` formulae (No Part.) of both benchmark sets, we compared the performance of UPMAX with some of the best solvers [1] in the MaxSAT Evaluation 2022 [2], such as MaxHS [13], UWrMaxSat-SCIP [46], CASHWMaxSAT-CorePlus [27], EvalMaxSAT [8], and MaxCDCL [29]. MaxHS is a MaxSAT solver based on an implicit hitting set approach. UWrMaxSat is an unsatisfiability-based solver using the OLL algorithm. These solvers can be seen as better versions than the RC2 and Hitman algorithms available in PySAT. CASH-WMaxSAT is developed from UWrMaxSat, EvalMaxSAT is based on the OLL algorithm, and MaxCDCL is an extension for MaxSAT of the CDCL algorithm [28], which combines Branch and Bound and clause learning.

Regarding the minimum sum coloring problem, MaxHS solved 873 instances, EvalMaxSAT solved 729, CASHWMaxSAT solved 993 (708 without SCIP), UWrMaxSat solved 994 (728 without SCIP), and MaxCDCL solved 995 instances. Note that solvers using Branch and Bound excel on these instances, and the performance of CASHWMaxSAT and UWrMaxSat deteriorates when SCIP is not used. Moreover, these results also show that partitioning improves less effective MaxSAT algorithms to become competitive with some solvers (e.g. UWrMaxSat), and outperform other solvers, e.g., MaxHS and EvalMaxSAT. Secondly, regarding the seating assignment problem, UWrMaxSat solved 580 instances, CASHWMaxSAT solved 585, MaxCDCL solved 593, MaxHS solved 643, and EvalMaxSAT solved 653 instances. When compared with our best results, note that table-based partitioning with the MSU3 algorithm can outperform all these solvers. Moreover, since partitioning can improve the performance of multiple MaxSAT algorithms and it is beneficial for implicit hitting set approaches like Hitman, it has the potential to further improve the performance of MaxHS.

Even though partitioning is not expected to improve the performance of MaxSAT solvers on all problem domains, there are many domains similar to the seating assignment and minimum sum coloring [2] for which partitioning can provide a significant performance boost in MaxSAT solving. Finally, we note this work opens new lines of research based on decoupling of MaxSAT solvers from the procedure that defines the partitions of MaxSAT formulae.

5   Conclusions

In this paper, we propose UPMAX, a new framework that decouples the partition generation from the MaxSAT solving. UPMAX allows the user to specify how to partition MaxSAT formulas with the proposed `pwcnf` format. With this format, the partitioning of MaxSAT instances can be done a priori to MaxSAT solving. Experimental results with two use cases with multiple algorithms show that partitioning can improve the performance of MaxSAT algorithms and allow them to solve more instances. UPMAX provides an extendable framework that can benefit (1) researchers on partitioning strategies, (2) solver developers with new MaxSAT algorithms that can leverage partition information, and (3) users that can benefit from additional information when modeling problems to MaxSAT.

---

[1] We did not modify any of these solvers since each solver has a large codebase with many optimizations, but these solvers could also use a partitioning approach.

[2] Check Alloy^{Max} [54] paper for UPMAX's results on other application domains.

──────── **References** ────────

1   `https://github.com/forge-lab/upmax`. Last accessed on 28th February 2023.

2   `https://maxsat-evaluations.github.io/2022/`. Last accessed on 28th February 2023.

3   C. Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The Community Structure of SAT Formulas. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 7317 of *LNCS*, pages 410–423. Springer, 2012.

4   Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving SAT-Based Weighted MaxSAT Solvers. In *Principles and Practice of Constraint Programming*, volume 7514 of *LNCS*, pages 86–101. Springer, 2012.

5   Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving WPM2 for (Weighted) Partial MaxSAT. In *Principles and Practice of Constraint Programming*, volume 8124 of *LNCS*, pages 117–132. Springer, 2013.

6   Carlos Ansótegui and Joel Gabàs. WPM3: an (in)complete algorithm for weighted partial maxsat. *Artificial Intelligence*, 250:37–57, 2017.

7   Josep Argelich and Felip Manyà. Exact max-sat solvers for over-constrained problems. *J. Heuristics*, 12(4-5):375–392, 2006.

8   Florent Avellaneda. A short description of the solver evalmaxsat. *MaxSAT Evaluation*, 8, 2020.

9   Jeremias Berg, Antti Hyttinen, and Matti Järvisalo. Applications of maxsat in data analysis. In Daniel Le Berre and Matti Järvisalo, editors, *International Workshop Pragmatics of SAT*, volume 59 of *EPiC Series in Computing*, pages 50–64. EasyChair, 2018.

10   V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*, 2008(10):P10008, 2008.

11   Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming*, volume 6876 of *LNCS*, pages 225–239. Springer, 2011. `doi:10.1007/978-3-642-23786-7_19`.

12   Jessica Davies and Fahiem Bacchus. Exploiting the Power of mip Solvers in maxsat. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *LNCS*, pages 166–181. Springer, 2013.

13   Jessica Davies and Fahiem Bacchus. Postponing Optimization to Speed Up MAXSAT Solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *LNCS*, pages 247–262. Springer, 2013.

14   Emir Demirovic, Nysret Musliu, and Felix Winter. Modeling and solving staff scheduling with partial weighted maxsat. *Annals OR*, 275(1):79–99, 2019. `doi:10.1007/s10479-017-2693-y`.

15   Niklas Eén and Niklas Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

16   Zhaohui Fu and Sharad Malik. On Solving the Partial MAX-SAT Problem. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 4121 of *LNCS*, pages 252–265. Springer, 2006.

17   Federico Heras, António Morgado, and João Marques-Silva. An empirical study of encodings for group maxsat. In Leila Kosseim and Diana Inkpen, editors, *Canadian Conference on Artificial Intelligence*, volume 7310 of *LNCS*, pages 85–96. Springer, 2012. `doi:10.1007/978-3-642-30353-1_8`.

18   Toshinori Hosokawa, Hiroshi Yamazaki, Kenichiro Misawa, Masayoshi Yoshimura, Yuki Hirama, and Masavuki Arai. A Low Capture Power Oriented X-filling Method Using Partial MaxSAT Iteratively. In *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 1–6. IEEE, 2019.

19   Alexey Ignatiev, António Morgado, and João Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *International Conference on Theory and Applications of Satisfiability Testing*, volume 10929 of *LNCS*, pages 428–437. Springer, 2018. `doi:10.1007/978-3-319-94144-8_26`.

20   Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. `doi:10.3233/SAT190116`.

21   Daniel Jackson. *Software Abstractions: Logic, language, and analysis.* MIT Press, 2006.

22   Mikoláš Janota, Inês Lynce, Vasco Manquinho, and Joao Marques-Silva. PackUp: Tools for Package Upgradability Solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):89–94, 2012.

23   Manu Jose and Rupak Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *Programming Language Design and Implementation*, pages 437–446. ACM, 2011.

24   Eunsuk Kang, Aleksandar Milicevic, and Daniel Jackson. Multi-representational security analysis. In *fse*, FSE 2016, pages 181–192. ACM, 2016.

25   Sarfraz Khurshid and Darko Marinov. Testera: Specification-based testing of java programs using SAT. *Autom. Softw. Eng.*, 11(4):403–434, 2004.

26   Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A Partial Max-SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012. `doi:10.3233/sat190091`.

27   Zhendong Lei, Shaowei Cai, Dongxu Wang, Yongrong Peng, Fei Geng, Dongdong Wan, Yiping Deng, and Pinyan Lu. Cashwmaxsat: Solver description. *MaxSAT Evaluation*, 2021:8, 2021.

28   Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining clause learning and branch and bound for maxsat. In Laurent D. Michel, editor, *Principles and Practice of Constraint Programming*, volume 210 of *LIPIcs*, pages 38:1–38:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.CP.2021.38`.

29   Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Boosting branch-and-bound maxsat solvers with clause learning. *AI Communications*, 35(2):131–151, 2022.

30   Ferney A. Maldonado-Lopez, Jaime Chavarriaga, and Yezid Donoso. Detecting network policy conflicts using alloy. In Yamine Aït Ameur and Klaus-Dieter Schewe, editors, *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, volume 8477 of *LNCS*, pages 314–317. Springer, 2014. `doi:10.1007/978-3-662-43652-3_31`.

31   Vasco Manquinho, Joao Marques-Silva, and Jordi Planes. Algorithms for Weighted Boolean Optimization. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *LNCS*, pages 495–508. Springer, 2009.

32   João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. Artif. Intell.*, 62(3-4):317–343, 2011. `doi:10.1007/s10472-011-9233-2`.

33   João Marques-Silva and Jordi Planes. On Using Unsatisfiability for Solving Maximum Satisfiability. *CoRR*, 2007. `arXiv:0712.1097`.

34   Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental Cardinality Constraints for MaxSAT. In *Principles and Practice of Constraint Programming*, volume 8656 of *LNCS*, pages 531–548. Springer, 2014.

35   Ruben Martins, Vasco Manquinho, and Inês Lynce. On partitioning for maximum satisfiability. In *European Conference on Artificial Intelligence*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 913–914. IOS Press, 2012.

36   Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: a Modular MaxSAT Solver. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 8561 of *LNCS*, pages 438–445. Springer, 2014.

37   Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Community-based partitioning for maxsat solving. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *LNCS*, pages 182–191. Springer, 2013.

38   Erick Moreno-Centeno and Richard M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Oper. Res.*, 61(2):453–468, 2013. `doi:10.1287/opre.1120.1139`.

39 A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

40 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-Guided MaxSAT with Soft Cardinality Constraints. In *Principles and Practice of Constraint Programming*, volume 8656 of *LNCS*, pages 564–573. Springer, 2014.

41 António Morgado, Alexey Ignatiev, and João Marques-Silva. MSCG: robust core-guided maxsat solving. *J. Satisf. Boolean Model. Comput.*, 9(1):129–134, 2014. `doi:10.3233/sat190105`.

42 Miguel Neves, Ruben Martins, Mikolás Janota, Inês Lynce, and Vasco Manquinho. Exploiting resolution-based representations for maxsat solving. In Marijn Heule and Sean A. Weaver, editors, *International Conference on Theory and Applications of Satisfiability Testing*, volume 9340 of *LNCS*, pages 272–286. Springer, 2015.

43 Pedro Orvalho, Vasco Manquinho, and Ruben Martins. Upmax: User partitioning for maxsat, 2023. `arXiv:2305.16191`.

44 Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic polynomial watchdog encoding for solving weighted maxsat. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 10929 of *LNCS*, pages 37–53. Springer, 2018.

45 Marek Piotrów. UWrMaxSat – a new MiniSat+ – based Solver in MaxSAT Evaluation 2019. *MaxSAT Evaluation 2019 : Solver and Benchmark Descriptions*, B-2019-2:11, 2019.

46 Marek Piotrów. UWrMaxSat: Efficient Solver for MaxSAT and Pseudo-Boolean Problems. In *International Conference on Tools with Artificial Intelligence*, pages 132–136. IEEE, 2020.

47 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid maxsat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 9710 of *LNCS*, pages 539–546. Springer, 2016.

48 Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Starexec: A cross-community infrastructure for logic solving. In *International Joint Conference on Automated Reasoning*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014.

49 Caroline Trippel, Daniel Lustig, and Margaret Martonosi. CheckMate: Automated synthesis of hardware exploits and security litmus tests. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 2018-Octob:947–960, 2018.

50 Allen Van Gelder. Variable independence and resolution paths for quantified boolean formulas. In *Principles and Practice of Constraint Programming*, volume 6876 of *LNCS*, pages 789–803. Springer, 2011. `doi:10.1007/978-3-642-23786-7_59`.

51 Rouven Walter, Christoph Zengler, and Wolfgang Küchlin. Applications of maxsat in automotive configuration. In *International Configuration Workshop*, volume 1128 of *CEUR Workshop Proceedings*, pages 21–28. CEUR-WS.org, 2013.

52 Robert A. Yates, Bertram Raphael, and Timothy P. Hart. Resolution graphs. *Artificial Intelligence*, 1(4):257–289, 1970. `doi:10.1016/0004-3702(70)90011-1`.

53 Aolong Zha, Miyuki Koshimura, and Hiroshi Fujita. *N*-level modulo-based CNF encodings of pseudo-boolean constraints for maxsat. *Constraints An Int. J.*, 24(2):133–161, 2019. `doi:10.1007/s10601-018-9299-0`.

54 Changjian Zhang, Ryan Wagner, Pedro Orvalho, David Garlan, Vasco Manquinho, Ruben Martins, and Eunsuk Kang. AlloyMax: bringing maximum satisfaction to relational specifications. In Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta, editors, *fse*, pages 155–167. ACM, 2021.

55 Lei Zhang and Fahiem Bacchus. MAXSAT heuristics for cost optimal planning. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2012. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5190`.