Parameterization of (Partial) Maximum Satisfiability above Matching in a Variable-Clause Graph

Vasily Alferov[1], **Ivan Bliznets**[2], **Kirill Brilliantov**[3]
[1]Independent Researcher
[2]University of Groningen
[3]St. Petersburg Department of Steklov Mathematical Institute of the RAS
vasily.v.alferov@gmail.com, iabliznets@gmail.com/i.bliznets@rug.nl, ki.br178@gmail.com

Abstract

In the paper, we study the Maximum Satisfiability and the Partial Maximum Satisfiability problems. Using Gallai–Edmonds decomposition, we significantly improve the upper bound for the Maximum Satisfiability problem parameterized above maximum matching in the variable-clause graph. Our algorithm operates with a runtime of $O^*(2.83^k)$, a substantial improvement compared to the previous approach requiring $O^*(4^k)$, where k denotes the relevant parameter. Moreover, this result immediately implies $O^*(1.14977^m)$ and $O^*(1.27895^m)$ time algorithms for the $(n, 3)$-MaxSAT and $(n, 4)$-MaxSAT where m is the overall number of clauses. These upper bounds improve prior-known upper bounds equal to $O^*(1.1554^m)$ and $O^*(1.2872^m)$. We also adapt the algorithm so that it can handle instances of Partial Maximum Satisfiability without losing performance in some cases. Note that this is somewhat surprising, as the existence of even one hard clause can significantly increase the hardness of a problem.

Introduction

The Maximum Satisfiability (MaxSAT) problem holds significant importance in various fields of artificial intelligence, computer science, mathematics, and engineering due to its theoretical relevance, algorithmic challenges, and practical applications. MaxSAT is a natural extension of the well-studied Boolean Satisfiability (SAT) problem. Investigating MaxSAT contributes to a deeper understanding of the complexity landscape of optimization and decision problems. The study of the Satisfiability and Maximum Satisfiability problems provides insights into the boundaries of what algorithms can achieve. The Satisfiability problem was one of the first problems for which NP-completeness was shown. For a detailed survey about SAT and MaxSAT problems, we refer to (Biere, Heule, and van Maaren 2021). A diverse array of strategies has been employed to address the inherent complexity of the MaxSAT problem. These encompass a spectrum of approaches, including randomized, approximation, exact, and parameterized algorithms.

After a prolonged period of silence, new results from an exact exponential point of view start to appear for MaxSAT again. At AAAI-21 (Alferov and Bliznets 2021) presented a new algorithm, introducing a novel approach with a runtime of $\mathcal{O}^*(1.0927^L)$[1], where L is the total number of literals within the input formula. Building upon this progress, at AAAI-2023 Brilliantov et al. published a subsequent enhancement that achieved an even more impressive upper bound of $\mathcal{O}^*(1.0911^L)$ (Brilliantov, Alferov, and Bliznets 2023). It is worth highlighting that prior to these results, the previous best-known upper bound dated back over two decades ago (Bansal and Raman 1999), yielding an upper bound of $\mathcal{O}^*(1.1057^L)$.

This trend of improvements also extends to another crucial measure of the input formulas: the total number of clauses. At AAAI-2019, Xu et al. introduced an algorithm with a time complexity of $\mathcal{O}^*(1.2989^m)$ (Xu et al. 2019), presenting a substantial leap in performance. Following this result, at IJCAI-2022, Xiao further refined the landscape by unveiling an algorithm with an improved time complexity of $\mathcal{O}^*(1.2886^m)$ (Xiao 2022). These advancements superseded the previously established best upper bound, which had remained unchallenged since 2004 (Chen and Kanj 2004).

The number of clauses that one is asked to satisfy in a given CNF formula is also a natural and well-studied measure. For this measure, the best-known algorithm was developed in (Chen, Xu, and Wang 2015), and the running time of the algorithm is $\mathcal{O}^*(1.325^{k'})$, where k' is the number of clauses that we need to satisfy (we note that generally this measure is denoted by k, in order to avoid ambiguity we denote this measure by k').

In this paper, we study the MaxSAT problem from parameterized point of view. Specifically, we study MaxSAT parameterized above lower bounds. The MaxSAT has several natural lower bounds computable in polynomial time. It is well known that there is a truth assignment that satisfies at least half of all clauses. If we consider an assignment π_1 that sets all variables to True, and assignment π_2 that sets all variables to False then at least one of them satisfies at least half of all clauses. Let us denote by ℓ_0 the number of all clauses satisfied by the assignment π_1. It is obvious, that optimal truth assignment satisfies at least ℓ_0 clauses. In order to compute the third lower bound we do the following. Let us construct a graph in which each vertex corresponds

[1]$\mathcal{O}^*(\cdot)$ supresses polynomial factors similarly as $\mathcal{O}(\cdot)$ supresses constant factors

either to a variable or to the clause from the input formula. We connect two vertices with an edge if and only if one of the vertices corresponds to a variable x, the other vertex corresponds to a clause C, and a variable x belongs to C. The constructed graph we call a variable-clause graph, and denote it by G_F (in future, if F is clear from the context we simply write G). Let us compute the maximum matching M in G_F and denote it by $\nu(G_F)$. It is easy to see that we can set values of variables so that we satisfy at least $\nu(G_F)$ clauses. Precisely, for each edge e in M, we pick a value of the variable corresponding to one of the endpoints of e so that the clause corresponding to the other endpoint is satisfied.

In parameterization above guarantee one is given an optimization problem, an integer k (parameter), a lower bound ℓ for an optimum value and the goal is to check if the optimum value is at least $\ell + k$. An interested reader can find more details on parameterization above guarantee in a recent survey (Gutin and Mnich 2022). Parameterization above lower bound $\frac{m}{2}$ was introduced in (Mahajan and Raman 1999). It was shown there that MAXSAT admits an FPT-algorithm under such parameterization. Moreover, they showed that the problem can be reduced to MAXSAT where the number of clauses that need to satisfy is used as a parameter without significant blow-up of the original parameter. In the case of a lower bound ℓ_0, it was shown (Belova and Bliznets 2020) that the problem is NP-complete even for $k = 1$. For a parameterization above lower bound $\nu(G_F)$ (Crowston et al. 2014) presented a deterministic algorithm with running time $\mathcal{O}^*(2e)^{2k+O(log^2 k)}$, and randomized algorithm with running time $\mathcal{O}^*(8^{k+O(\sqrt{k})})$. Later, advances for $(n-k)$-SET COVER problem lead to a $\mathcal{O}^*(4^k)$ algorithm for this parameterization (Basavaraju et al. 2016). In this paper we further improve algorithm for this parameterization. Note that the measure k' (number of clauses that need to satisfy) also can be considered as parameterization above guarantee where the lower bound is 0.

Our Results

We employ Gallai-Edmonds decomposition and significantly improve upper bound for $(n-k)$-SET COVER. The new algorithm works in $2^{\frac{3k}{2}}$ time improving previous $O(4^k)$ time algorithm. This result imply $2^{\frac{3k}{2}}$ time algorithm for MAXSAT parameterized above maximum matching in a variable-clause graph. Besides we adapt reduction from MAXSAT to $(n-k)$-SET COVER so that it can handle hard clauses. Formally, we prove that PARTIAL MAXIMUM SATISFIABILITY above maximum matching can be solved in $2^{\frac{3k}{2}}$ time if all hard clauses has length at most 2, and in time $\mathcal{O}^*(2^{\frac{3k}{2}} + 2^k \cdot 1.12226^h)$ if the number of hard clauses is h. Moreover, our result imply $\mathcal{O}^*(1.14977^{k'})$, $\mathcal{O}^*(1.14977^m)$ (here m is overall number of clauses in the input formula and k' is the number of clauses that we need to satisfy) time algorithms for MAXSAT restricted to instances where each variable appears at most three times and an $\mathcal{O}^*(1.27895^{k'})$, $\mathcal{O}^*(1.27895^m)$ time algorithms for MAXSAT instances where each variable appears at most four times. Note that previous best-known algorithms for these special cases were $\mathcal{O}^*(1.1554^k)$, $\mathcal{O}^*(1.1554^m)$ (Brilliantov, Alferov, and Bliznets 2023), $\mathcal{O}^*(1.2872^m)$ (Xiao 2022), $\mathcal{O}^*(1.2989^k)$ (Brilliantov, Alferov, and Bliznets 2023). We complement our theoretical findings with computational experiments. Note that the goal of the algorithm to be efficient when parameter k is small (such instances can arise during an execution of branching algorithm). If this is not the case then even simple brute-force algorithm outperform our algorithm. We compare performance of our solver with the state-of-the-art open source MaxSAT solvers on type of instances for which our algorithm was designed. It is obvious that on general instances our algorithm will be less efficient as it does not contain sophisticated heuristics that are very useful on practice, however, lack proved guarantee on the running time. Through this testing, our objective was to determine whether there are scenarios in which our algorithm demonstrates superior performance compared to the currently most competitive solvers. Note that on practice algorithms with the best proven guarantee can be significantly slower than algorithms based on heuristics without proven efficiency. Turns out that in this setting our algorithm outperforms other solvers.

Preliminaries

In the paper, we assume that a reader is familiar with concepts such as boolean variables, literals, and clauses. A variable x is an (i,j)-variable in the formula F if literal x and literal $\overline{x}$ appear i times and j times in F, respectively. For a variable x, we call literal x positive and literal $\overline{x}$ negative. Throughout the whole paper we assume that for each variable x literal x appears at least the same number of times as literal $\overline{x}$. Note that if some variable x in a formula does not satisfy the property, we can simply replace all literals x with $\overline{x'}$, and all literals $\overline{x}$ with x'. We call clause positive if it contains only positive literals. For convenience instead of clause $(x_1 \vee x_2 \vee \cdots \vee x_k)$ we often write slightly abusing notation simply $x_1 x_2 \dots x_k$. Interested readers may find details about CNF formula, satisfiability, the MAXSAT problem and further relevant information in (Marek 2009). We also assume that the reader is familiar with basic notions of parameterized algorithms, and branching algorithms. Relevant details for this two subjects can be found in (Cygan et al. 2015) and (Fomin and Kratsch 2010) respectively. In the paper, we study the MAXIMUM SATISFIABILITY problem, for short MAXSAT. In the problem, one is given a formula F in CNF, integer k' and the goal is to check if at least k' clauses can be satisfied in F. If we restrict our inputs to formulas where each variable appears at most s times we call such problem (n,s)-MAXSAT. We also consider PARTIAL MAXIMUM SATISFIABILITY problem, in this problem comparing with MAXSAT we additionally have hard clauses that must be satisfied, i.e. we need to check if there is an assignment that satisfies all hard clauses and at least k' clauses. Main focus of our paper is the MAXIMUM SATISFIABILITY problem parameterized above matching. In this problem, we are given a formula F, an integer k and our goal is to check if we can satisfy at least $\nu(G_F)+k$ clauses in F, where $\nu(G_F)$ is the size of the maximum matching in a variable-clause

graph G_F, defined in the introduction. In future, sometimes we use shorthand $\nu(F)$ instead of $\nu(G_F)$. For just presented problem we use shorthand $(\nu(F)+k)$-SAT. Similarly, if additionally we have hard clauses, we use shorthand Partial $(\nu(F)+k)$-SAT. Throughout the whole paper, we denote the overall number of clauses in the input formula by m and the number of variables by n. If we apply reduction rule to an instance of Partial $(\nu(F)+k)$-SAT we say that initial instance has a formula F and a new instance has a formula F'. Moreover, a parameter of the initial instance is k and a parameter of a new instance is k'. As the key of our advances for MAXSAT is improvement for the $(n-k)$-SET COVER problem. We define the latter problem here. In the $(n-k)$-SET COVER problem we are given a universe $\mathcal{U} = \{1, 2, \ldots, n\}$, a family $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$ such that $S_i \subset \mathcal{U}$ for any $1 \leq i \leq m$, an integer k, and the goal is to check if there are $i_1, i_2, \ldots i_{n-k}$ such that $S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_{n-k}} = \mathcal{U}$.

We also employ theorems about graphs and the Satisfiability problem listed below.

Theorem 1 (Szeider (2004))**.** *If a formula F contains n variables, $n+1$ clause and $\nu(F) = n$ then we can check in time $\mathcal{O}(n^3)$ if F is satisfiable or not.*

Theorem 2 (generalization of Hall's lemma)**.** *Let $d \geq 0$, G – bipartite graph with parts A, B, such that for any $X \subseteq A$ we have $|N(X)| \geq |X| - d$. If M is a maximum matching in G then $|M| \geq |A| - d$.*

Theorem 3 (Hopcroft and Karp (1973))**.** *Let G be a bipartite graph with bipartition V_1 and V_2, on n vertices and m edges. Then we can find a maximum matching of G in time $O(m\sqrt{n})$. Furthermore, in time $O(m\sqrt{n})$ either we can find a matching saturating V_1 or an inclusion-wise minimal set $X \subset V_1$ such that $|N(X)| < |X|$.*

Theorem 4 (Gallai–Edmonds decomposition)**.** *For any graph G, we can split in polynomial time its vertices into three sets C, B, D such that for any maximum matching M in G, we have: (i) there is no edge going from the set C to the set D; (ii) all connected components of $G[C]$ have an even number of vertices and contain perfect matching; (iii) all connected components in $G[D]$ consist of an odd number of vertices and are factor-critical; (iv) matching M induces a perfect matching in C, an almost perfect matching in each connected component of $G[D]$ (almost perfect matching cover all vertices except one vertex); (v) matching M to each vertex from the set B assign its unique connected component of $G[D]$ with which the vertex is connected by an edge from M.*

Theorem 5 (Fomin and Kratsch (2010), Corollary 3.25)**.** $\sum_{j=0}^{n} \sum_{k=j}^{\min(2j,n)} \binom{n-j}{k-j} = O(poly(n) \cdot \varphi^n)$*, here and later* φ *denotes* $\frac{1}{2}(\sqrt{5}+1)$.

Due to space constraints all proofs of Section "Partial Maximum Satisfiability" are deferred to the full version.

Partial Maximum Satisfiability

Crowston et al. (2014) essentially proved the following theorem.

Theorem 6 (Crowston et al. (2014))**.** *If $(n-k)$-SET COVER admits an algorithm with running time $f(k) = \Omega(2^k)$ then $(\nu(F)+k)$-SAT admits $O^*(f(k))$-time algorithm.*

Inspired by Reduction and Branching rules from (Crowston et al. 2014), we construct adapted Reduction and Branching rules for Partial $(\nu(F)+k)$-SAT such that eventually for some special cases of Partial $(\nu(F)+k)$-SAT we obtain the same upper bound as for $(\nu(F)+k)$-SAT. Note that existence of such modification is somewhat surprising as generally existence of hard clauses makes problem significantly more difficult from theoretical point of view. In the following example we show that the presence of only one short hard clause significantly changes computational complexity of a problem. Consider a family of formulas that have the following type $xC_1 \wedge xC_2 \wedge \cdots \wedge xC_m \wedge x \wedge \overline{x}$. It is obvious that we can satisfy maximum $m+1$ clause in such formulas, and it is enough to set $x = 1$ to do this. Hence, MAXSAT is solvable in polynomial time on this type of instances. It is enough to check that an input instance match the format and count the overall number of clauses. However, if we consider PARTIAL MAXSAT on this type of instances and mark $\overline{x}$ as a hard clause then essentially we have to solve MAXSAT on the formula $C_1 \wedge \cdots \wedge C_m$ where there is no restriction on clauses $C_1, C_2, \ldots, C_m$ at all. Therefore PARTIAL MAXSAT restricted to these instances cannot be solved using $2^{o(m)}$ time assuming the Exponential Time Hypothesis. Hence, this example provides us with an exponential gap between the running time of algorithms for MAXSAT and the running time for PARTIAL MAXSAT assuming ETH.

We prove the following theorem in this section.

Theorem 7. *If $(n-k)$-SET COVER is solvable in $\mathcal{O}^*(f(k))$ time where $f(k) \geq 2^k$ then Partial $(\nu(F)+k)$-SAT admits a $\mathcal{O}^*(f(k))$ time algorithm on instances where all hard clauses have length at most 2. Moreover, under the same assumption Partial $(\nu(F)+k)$-SAT admits $\mathcal{O}^*(f(k) + 2^k \cdot 1.12226^h)$ time algorithm on formulas with h hard clauses.*

Before we present proof of the above theorem, we provide several Reduction rules and prove their correctness. If Reduction rule R is applied to an instance (F, k) and produces an instance (F', k') then we say that R is correct if: (i) (F, k) is a YES-instance if and only if (F', k') is a YES-instance; (ii) $k' \leq k$; (iii) number of hard clauses in F' is not larger then number of hard clauses in F; (iv) if all hard clauses in F have length at most 2 then all hard clauses in F' also have length at most 2. Similarly, for correctness of a Branching rule that produces two instances $(F_1, k_1), (F_2, k_2)$ we require: (i) (F, k) is a YES-instance if and only if (F_1, k_1) or (F_2, k_2) are YES-instance instances; (ii) $k_1, k_2 < k$; (iii)the number of hard clauses in F_1, F_2 is not larger then the number of hard clauses in F; (iv) if all hard clauses in F have length at most 2 then all hard clauses in F_1, F_2 have length at most 2.

Lemma 1. *Let M be a maximum matching in a variable-clause graph of a formula F, and let π_M be a satisfying assignment that correspond to this matching (clauses from M are satisfied by variables from M). If π_H is a satisfying assignment that satisfies all hard clauses in F then in polynomial time we can construct a truth assignment π such that*

π satisfies all hard clauses and overall number of satisfied clauses is at least $\nu(F)$.

Previous lemma implies our first Reduction rule.

Reduction rule 1. *If $k \leq 0$, and all hard clauses have length at most 2 then solve the instance in polynomial time. Otherwise, solve the instance in time $\mathcal{O}^*(1.2226^h)$ where h is the total number of hard clauses in F.*

Reduction rule 2. *If formula contains a clause $C = l\bar{l}C'$ for some literal l then delete the clause. Decrease the parameter by one if the matching size in the variable-clause graph did not change, and leave the parameter unchanged if the matching size decreased by one.*

Reduction rule 3. *If there is a literal l such that literal $\bar{l}$ does not appear in F then set $l = 1$ and recompute the value of the parameter.*

Reduction rule 4. *If there is variable x such that F contains hard clauses x and $\overline{x}$ then return* NO.

Reduction rule 5. *If formula contains hard clause l where l is some literal, then set $l = 1$ and recompute the value of the parameter.*

Reduction rule 6 (Resolution)**.** *If F contains a $(1, 1)$-variable x with clauses $xC, \overline{x}D$ then we replace these two clauses with clause CD. Moreover, we mark the new clause as a hard clause if and only if both clauses $xC, \overline{x}D$ were marked as hard. After that we recompute the value of the parameter depending on the change of the maximum matching.*

We call a bipartite graph G with parts A, B *t-expanding* if for each $X \subseteq A, X \neq \emptyset$ we have $|N_G(X)| \geq |X| + t$. Similarly, we call a formula F *t-expanding* if the corresponding variable-graph G_F is t-expanding.

Reduction rule 7. *If the variable-clause graph G of the formula F is not 1-expanding i.e. there is a set of variables C such that $|N_G(C)| \leq |C|$ then find such inclusion minimal set C and assign values to the variables from C so that all clauses from $|N_G(C)|$ are satisfied.*

From now on we can assume that for the input formula F the variable-clause graph is 1-expanding.

If variable-clause graph G of the formula F is not 2-expanding then we find a set of variables S such that $|N_G(S)| = |S| + 1$. We can find such set in polynomial time by Lemma 8 in (Crowston et al. 2014). Denote by F_S a formula that we get if we delete all variables from F except for the variables from the set S. We apply Theorem 1 to the formula F_S and check if F_S is satisfiable.

Reduction rule 8. *If F_S is satisfiable, then we assign values to the variables from S such that all clauses in $N(S)$ are satisfied.*

Reduction rule 9. *If F_S is not satisfiable we delete from the formula F the set of clauses corresponding to $N_G(S)$ and add a new clause which contains all literals from all clauses of $N_G(S)$ except literals of variables from the set S. Let us call the new clauses C. We mark C as hard if and only if all clauses in $N_G(S)$ are hard.*

After application of all Reduction Rules listed above we apply branching rules. Before we present the branching rules, we prove the following lemma.

Lemma 2. *If formula F has 2-expanding variable-clause graph and x is $(a, 1^+)$-variable with $a > 1$ then $\nu(F_{x=1}) \geq \nu(F) - a + 1$.*

Branching rule 1. *If $k > 0$ and there is $(2^+, 2^+)$-variable x in F then we consider two branchings $x = 0$ and $x = 1$. In both branchings we recompute the value of the parameter and it drops at least by one.*

If Branching rule 1 is not applicable, we try to apply the following Branching rule.

Branching rule 2. *If the formula F contains a clause $\overline{x}\overline{y}C$ then consider two branchings $F[x = 1]$ and $F[y = 1]$. In both branchings we recompute the value of the parameter and it drops at least by one.*

Note that if none of the above rules is applicable, then negative literals appear exactly once (as for each variable x literal x appears at least the same number of times as literal $\overline{x}$), and each clause contains at most one negative literal. If F has this type, then we assign to it the following directed graph D_F: for each variable x we introduce a vertex v_x, for each clause $\overline{x}y_1 \dots y_k$ we introduce edges $v_x v_{y_1}, \dots, v_x v_{y_k}$.

Reduction rule 10. *If D_F contains a cycle $v_{z_0} v_{z_1} \dots v_{z_\ell}$ then set $z_0 = z_1 = \dots = z_\ell = 1$.*

Lemma 3. *If none of the above rules is applicable to the formula F, then Partial $(\nu(F) + k)$-SAT and $(\nu(F) + k)$-SAT are equivalent on the formula F. Formally, if π satisfies k clauses in F then there is an assignment π' that satisfies k clauses in F and all hard clauses in F. Moreover, given π we can find π' in polynomial time.*

The above lemma helps us to reduce Partial $(\nu(F) + k)$-SAT to $(\nu(F) + k)$-SAT. Now, we can apply all reductions from (Crowston et al. 2014) without any adaptation. The following Reduction rule was presented in (Crowston et al. 2014) as Lemma 11.

Reduction rule 11 (Crowston et al.)**.** *If there is a variable x such that F contains clauses $xC_1, \dots, xC_i, \overline{x}D$ (this is an exhaustive list of all clauses containing variable x) and D is non-empty. Then replace these clauses with $xC_1D, \dots, xC_iD, \overline{x}$ correspondingly.*

After exhaustive application of all rules above all variables are $(i, 1)$-variables and for each variable x there is a clause $\overline{x}$ and it is the only occurrence of the literal $\overline{x}$.

Now, we show how to transform our current instance of $(\nu(F)+k)$-SAT into $(n-k)$-SET COVER. First of all, note that for the current instance there is an optimal matching that satisfies all clauses with positive literals. Indeed all clauses with negative literals contain only one literal i.e. can be expressed as $\overline{x}$. Moreover, all such clauses are not hard as otherwise Reduction rule 5 is applicable. So, if clauses with positive literal x is not satisfied then we can flip value of x. In this case we satisfy at least one clause and at most one clause become unsatisfiable and this is not a hard clause. Therefore, from now on we are looking for an assignment that satisfies all positive clauses. Note that matching in the

variable-clause graph has size n as we simply can match each variable x_i with the clause $\overline{x_i}$. Our reduction works in the following way: (i) for each positive clause C_j we create an element j; (ii) for each variable x_i we create a set S_i, the set S_i contains an element j if and only if $x_i \in C_j$. So, our instance of SET COVER has universe $\mathcal{U}$ of size $n' = m - n$, and a family of covering sets $\mathcal{F}$ of size $m' = n$. Note that we can satisfy $\nu(F)+k = n+k$ clauses in F if and only if there is a covering containing at most $n'-k = m-n-k$ sets in the constructed SET COVER instance. Indeed, if π is an assignment satisfying $n+k$ clauses and all positive clauses, then at least $(n+k)-(m-n) = 2n+k-m$ variables are set to 0. Hence, there is at most $n-(2n+k-m) = m-n-k = n'-k$ variables that are set 1, and family $\{S_i|\pi(x_i) = 1\}$ covers the whole $\mathcal{U}$. Moreover, if sets $S_{i_1}, S_{i_2}, \ldots, S_{i_p}$ cover $\mathcal{U}$, then an assignment with $x_{i_1} = x_{i_2} = \cdots = x_{i_p} = 1$ and all other variables set to 0 satisfy $C_1, \ldots, C_{m-n}$ and $n - p$ clauses from $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$. Hence, covering with $p = n' - (n' - p) = n' - k'$ sets was converted into a assignment satisfying $m-n+(n-p) = \nu(F)+n'-p = \nu(F)+k'$. Similar reduction was described in (Crowston et al. 2014)), so more details can he found there. Hence, in order to solve Partial $(\nu(F) + k)$-SAT it is enough to solve $(n - k)$-SET COVER.

In order to transform Partial $(\nu(F) + k)$-SAT to $(n - k)$-SET COVER we use Reduction rules and Branching rules. Note that Reduction rules can by applied in polynomial time. All of the Branching rules have two cases and parameter decreases by 1 in both cases. Hence, all of the branching factors are 2. After exhaustive application of the reduction and branching rules either we obtain an instance with $k = 0$ or we reduce the problem to $(n - k)$-SET COVER. Hence, we proved Theorem 7.

Algorithm for $(n - k)$-SET COVER

In this section we present $\mathcal{O}^*(2^{\frac{3}{2}k})$ time algorithm for $(n-k)$-SET COVER. By Theorem 7, this result implies that $(\nu(F) + k)$-SAT admits an $\mathcal{O}^*(2^{\frac{3}{2}k})$ time algorithm.

Theorem 8. $(n - k)$-SET COVER *admits an* $\mathcal{O}^*(2^{\frac{3}{2}k})$ *time algorithm.*

Proof. First of all our algorithm checks if each element is contained at least in one set. If it is not the case then we can immediately output NO. After that, our algorithm execute a simple greedy procedure. The procedure goes through all sets in arbitrary order and picks a set S' into cover if the set S' cover at least 3 new elements. The greedy procedure stops when each of the remaining sets covers at most two new elements. We denote by A the subset of all covered elements at the greedy step, and by R we denote the set of all sets that were picked during the procedure.

Note that if $A \geq \frac{3}{2}k$ then there is a set cover of size at most $n - k$. Indeed, there are $n - |A|$ uncovered elements, it is obvious that they can be covered by $n - |A|$ sets (roughly speaking for each element we have a special set covering it). Hence, our covering contains at most $|R| + n - |A|$ sets. Recall that each set in R covers at least 3 new elements, hence, we have that $R \leq \frac{1}{3}|A|$. Hence, our covering is of size at most $|R| + n - |A| \leq n - \frac{2}{3}|A| \leq n - k$.

It means that from now on we can assume that $|A| \leq \frac{3}{2}k$, and $|R| \leq \frac{1}{2}k$.

Now we consider elements from the set $V = \mathcal{U} \setminus A$. By the definition of the set A we have that each set from the family $\mathcal{F}$ covers at most two elements in V. We construct graph G based on V and $\mathcal{F}$. Each vertex in G corresponds to an element from the set V. Vertices u, v in G are connected by an edge if there is a $W \in \mathcal{F}$ that contains both elements from V that correspond to vertices u, v.

Denote by M a maximum matching in the graph G. Note that if $|M| \geq k - |A| + |R|$ then our instance of $(n-k)$-SET COVER is a YES-instance. Indeed, consider a covering that contains all sets from R, sets that correspond to edges from the matching M, and for each yet uncovered element, we use a new set. Note that the number of elements not covered by sets from R and sets corresponding to edges from M is exactly $n - |A| - 2|M|$. So, overall constructed covering contains $|R| + |M| + (n - |A| - 2|M|) = n + |R| - |A| - |M| \leq n + |R| - |A| - (k - |A| + |R|) = n - k$ sets. Therefore, from now on without loss of generality, we can assume that $|M| < k - |A| + |R|$.

Now, for the graph G, we find a set B described in Gallai-Edmonds decomposition (Theorem 4). Recall that all edges in M cover at most one vertex from B and all vertices in B are covered by M. Hence, $|B| \leq |M|$. Let H be a connected component in the graph $G \setminus B$. We know that M covers almost all vertices in H (at most one vertex is not covered) solely by edges fully contained in H. Hence, $|V(H)| \leq 2(|M| - |B|) + 1 \leq 2(k - |A| + |R| - |B|) + 1$.

Denote by $A' = A \cup B$, $V' = V \setminus B$. We order vertices in V' such that vertices of each connected component in $G[V']$ are consecutive in the ordering. Let us denote the ordering by π.

Now our algorithm employs dynamic programming. We fill in some cells in the table dp indexed by a subset $X \subseteq A'$ and a subset $Y \subseteq V'$. Our goal is to store in the cell $dp(X, Y)$ a minimum number of sets from $\mathcal{F}$ that are needed to cover the set $X \cup Y$. We note that we compute values not in all cells of the table dp and we do not create a whole table in a memory to store the data. We compute the values only in those cells that will be required to compute the value $dp(A', V')$.

We perform a computation of values in the table $dp(\cdot, \cdot)$ in the following way. First of all we initialize $dp(\emptyset, \emptyset) = 0$, undefined values we treat as $+\infty$. Knowing the value in the cell $dp(X, Y)$ we take the smallest element v (smallest with respect to the ordering π) in $V' \setminus Y$. For each subset $S \in \mathcal{F}$ such that $v \in S$ we update a value in the cell $dp(X \cup (S \cap A'), Y \cup (S \cap V'))$ with the value $\min\{dp(X \cup (S \cap A'), Y \cup (S \cap V')), dp(X, Y) + 1\}$. After that we go to the next cell $dp(X, Y)$ such that: (i) value $dp(X, Y)$ was already updated; (ii) value of minimum element in $V' \setminus Y$ is the smallest among all potential pairs $dp(X, Y)$. Finally, when we exhaust all elements from V' we fix some ordering σ on $\mathcal{F}$, and in order σ for all $S \in \mathcal{F}$ and all $X \subseteq A'$ we update answers in cells $dp(X \cup (S \cap A'), V')$ with value

$dp(X, V')+1$, i,e. we run simple dynamic programming for a set cover with the second coordinate being fixed with the value V'.

Claim 1. *After execution of the algorithm the cell $dp(A', V')$ stores a minimum number of sets from $\mathcal{F}$ required to cover the set $\mathcal{U} = A' \cup V'$.*

Proof. First of all note that the value of the smallest element uncovered by a set Y never decreases. Indeed, staying in the cell $dp(X, Y)$ we update values only in cells (X', Y') such that $Y \subsetneq Y'$.

Let $\mathcal{A} \subseteq \mathcal{F}$ be an optimum answer. We order sets in $\mathcal{A}$ in the following way: (i) let $S'_1 \in \mathcal{A}$ be an arbitrary set containing the smallest element in V'; (ii) let $S'_2 \in \mathcal{A}$ be an arbitrary set containing the smallest element in $V' \setminus S'_1$; (iii) let $S'_2 \in \mathcal{A}$ be arbitrary set containing the smallest element in $V' \setminus (S'_1 \cup S'_2)$; (iv) and so on until we cover all elements in V'; (v) after that we list the rest of sets from $\mathcal{A}$ in order σ. We assume that we obtain the following ordering $S'_1, S'_2, \ldots, S'_{|\mathcal{A}|}$. Denote by $T_i = \bigcup_{j=1}^{i} S'_j$.

We prove that $dp(T_i \cap A', T_i \cap V') \leq i$ by induction. It is obvious for $i = 0$, $dp(\emptyset, \emptyset) = 0 = i$. If $V' \not\subseteq T_{i-1}$, then by definition of S'_i staying in cell $(T_{i-1} \cap A', T_{i-1} \cap V')$, we update $dp((T_{i-1} \cup S_i) \cap A', (T_{i-1} \cup S_i) \cap V') = (T_i \cap A', T_i \cap V')$ with the value $dp(T_{i-1} \cap A', T_{i-1} \cap V') + 1 \leq i$. If $V' \subseteq T_{i-1}$, then being in cell $dp(T_{i-1} \cap A', V')$ we update cell $dp((T_{i-1} \cup S'_i) \cap A', V') = (T_i, V')$ with the value $dp(T_i \cap A', V') + 1$. Moreover, at this point we know that $dp(T_{i-1} \cap A', V') \leq i - 1$.

Therefore, $dp(A', V') \leq |\mathcal{A}|$. Note, that we always can construct a covering with at most $dp(A', V')$ sets simply analyzing which sets we used in dynamic programming step. Hence, $dp(A', V') = |\mathcal{A}|$. □

Essentially above claim proves the correctness of the algorithm. It is left to prove an upper bound on the running time.

First of all, note that running time of the last phase of the dynamic programming when the second coordinate Y equals to V' is $\mathcal{O}^*(2^{|A'|})$ (essentially this is standard dynamic programming for the SET COVER problem).

Recall that π order all vertices of V' such that connected components of $G[V']$ induce a consecutive blocks in the π. So, essentially π also induce an ordering on connected components of $G[V']$. Let us denote connected components of $G[V']$ by $H_1, H_2, \ldots, H_r$. We can assume that all vertices in H_i are smaller than all vertices in H_j if $i < j$.

Note that the running time of our algorithm up to polynomial time is equal to the number of updated cells in the table $dp(\cdot, \cdot)$. Let (X, Y) be one of such cells. If the smallest vertex outside of Y belongs to H_i then Y contains only vertices from connected components $H_1, H_2, \ldots H_i$. Moreover, Y contains all vertices from $H_1, H_2, \ldots H_{i-1}$. Let us denote by $f(H_i)$ a number of all $Y' \subseteq H_i$ such that there is X and the cell $(X, Y' \cup \bigcup_{j=1}^{i-1} H_j)$ was updated. It is easy to see that the overall number of updated cells is at most $2^{|A'|}(f(H_1) + f(H_2) + \cdots + f(H_r)) \leq r2^{|A'|} \cdot \max_i f(H_i) = \mathcal{O}^*(2^{|A'|} \max_i f(H_i))$. Now we provide an upper bound on the number $f(H_i)$. Let π defines the following ordering $v^i_1, v^i_2, \ldots v^i_{|H_i|}$ on the vertices of the connected component H_i. Note that if $Y' \subseteq H_i$ with described properties, and $\{v^i_1, v^i_2, \ldots, v^i_j\} \subseteq Y'$, $v^i_{j+1} \notin Y'$ then $|Y'| \leq 2j$. Indeed we always increase the second coordinate by some set with at most two elements and one of this elements is currently the smallest uncovered element. So, each Y_i with cell $(X, Y' \cup \bigcup_{j=1}^{i-1} H_j)$ being updated can be constructed in the following way: fix some j, take first j elements from the H_i, and take at most j elements from the set $H_i \setminus \{v^i_1, v^i_2, \ldots, v^i_j\}$. Therefore the total number of all such Y' is at most: $\sum_{j=0}^{|H_i|} \sum_{k=j}^{\min(2j,|H_i|)} \binom{|H_i|-j}{k-j} \leq \sum_{j=0}^{\frac{|H_i|}{2}} \sum_{k=j}^{\min(2j,|H_i|)} \binom{|H_i|-j}{k-j} + \sum_{j=\frac{|H_i|}{2}}^{|H_i|} \sum_{k=j}^{|H_i|} \binom{|H_i|-j}{k-j}$.

Note that $\sum_{j=\frac{|H_i|}{2}}^{|H_i|} \sum_{k=j}^{|H_i|} \binom{|H_i|-j}{k-j} \leq |H_i| 2^{\frac{|H_i|}{2}}$. By Theorem 5 we have that $\sum_{j=0}^{\frac{|H_i|}{2}} \sum_{k=j}^{\min(2j,|H_i|)} \binom{|H_i|-j}{k-j} \leq \mathcal{O}^*(\varphi^{|H_i|})$. Since, $\sqrt{2} < \varphi$ we have that $f(H_i) \leq \mathcal{O}^*(\varphi^{|H_i|})$. Therefore, overall running time of the algorithm is $\mathcal{O}^*(\varphi^{\max_i |H_i|} 2^{|A'|})$. It is left to bound the expression in terms of k. In order to do that we compute logarithm base 2 of this expression $\log_2 \left(\varphi^{\max_i |H_i|} 2^{|A'|}\right) = |A'| + (\max_i |H_i|) \log_2 \varphi$.

Recall that $|A'| = |A| + |B|$, and $|H_i| \leq 2(k - |A| + |R| - |B|) + 1$. Hence, the above expression is bounded by $|A| + |B| + (2(k - |A| + |R| - |B| + 1) \log_2 \varphi =$
$|A| \cdot (1 - 2\log_2 \varphi) + |B| \cdot (1 - 2\log_2 \varphi) + |R| \cdot 2\log_2 \varphi +$
$+ k \cdot 2\log_2 \varphi + \log_2 \varphi$.

Since, $1 - 2\log_2 \varphi < 0$, the expression is not larger than $|A| \cdot (1 - 2\log_2 \varphi) + |R| \cdot 2\log_2 \varphi + k \cdot 2\log_2 \varphi + \log_2 \varphi$.

Since, $|A| \geq 3|R|$ and $1 - 2\log_2 \varphi < 0$ the above expression can be bounded by: $|R| \cdot 3(1 - 2\log_2 \varphi) + |R| \cdot 2\log_2 \varphi + k \cdot 2\log_2 \varphi + \log_2 \varphi =$
$= |R| \cdot (3 - 4\log_2 \varphi) + k \cdot 2\log_2 \varphi + \log_2 \varphi \leq$
$\leq \frac{k}{2} \cdot (3 - 4\log_2 \varphi) + k \cdot 2\log_2 \varphi + \log_2 \varphi = \frac{3k}{2} + \log_2 \varphi$.

Therefore the running time of our algorithm is at most $\mathcal{O}^*(2^{\frac{3}{2}k})$ and we proved the desired claim. □

Natural Parameterization

In order to obtain an improvement for $(n, 3)$-MAXSAT, and $(n, 4)$-MAXSAT we employ the following theorem.

Theorem 9. *(Belova and Bliznets 2020; Brilliantov, Alferov, and Bliznets 2023) Assume that MAXSAT parameterized above matching can be solved in $O^*(c_1^k)$ time, (n, s)-MAXSAT can be solved in $O^*(c_2^n)$ and $c = c_2^{\frac{\log c_1}{\log c_1 + \log c_2}}$. In this case, for CNF formulas where each variable appears at most s times we can check if at least k' clauses are satisfiable in $O^*(c^{k'})$ time.*

Using Theorems 6, 8, 9, we get the following result.

Theorem 10. $(n, 3)$-MAXSAT *admits* $O^*(1.14977^{k'})$, $O^*(1.14977^m)$ *time algorithms.* $(n, 4)$-MAXSAT *can be solved in* $O^*(1.27895^{k'})$ *running time. Here, k' denotes number of clauses that we need to satisfy.*

This theorem improves previous known upper bounds for $(n, 3)$-MaxSAT which were $O^*(1.1554^{k'})$ and $O^*(1.1554^m)$ (Brilliantov, Alferov, and Bliznets 2023). It also improves upper bounds $O^*(1.2872^m)$ (Xiao 2022), $O^*(1.3248^{k'})$ (Chen, Xu, and Wang 2017) which follows for $(n, 4)$-MaxSAT from algorithms for general MaxSAT.

Experiments

In this section, we present the experiments conducted to evaluate the performance of our implemented algorithm in C++. The primary objective of these experiments was to assess the algorithm's efficiency in solving various instances of the MaxSAT problem for which solver was designed. To do this we consider performance of our algorithm and open-source solvers on specially generated instances. More information about implementation and environment details can be found in the full version.

Generated Tests

The generator used to test the capabilities of the algorithm accepts three parameters: a, b, and k. It creates a formula with $\nu(F) = ab$ and at most $ab + k$ satisfiable clauses.

There are ab variables in the test: $x_{1,1}, \ldots, x_{1,b}, \ldots, x_{2,1}, \ldots, x_{2,b}, \ldots, x_{a,1}, \ldots, x_{a,b}$. For each variable $x_{i,j}$, the clause $\overline{x_{i,j}}$ is added. For each $1 \leqslant i \leqslant a$, the clause $x_{i,1} \ldots x_{i,b}$ is added. Furthermore, we create k clauses more, and for each variable, its positive literal is added to a random subset of these clauses.

Note that we can match each variable $x_{i,j}$ with the clause $\overline{x_{i,j}}$, so $\nu(F) = ab$. Moreover, we claim that in this assignments we can satisfy at most $ab + k$ clauses. Indeed, let the optimal satisfying assignment satisfy c clauses of the form $x_{i,1} \ldots x_{i,b}$. Then, out of the clauses $\overline{x_{i,j}}$, at least c must be unsatisfied. Therefore, the optimal satisfying assignment satisfies at most $k + c + (ab - c) = k + ab$ clauses.

Baselines

We considered the following solvers (all participants of MaxSat Evaluations 2022): `cash-w-maxsat-coreplus` (Lei et al. 2022), `cgss` (Ihalainen, Berg, and J"arvisalo 2021), `eval-max-sat` (Avellaneda, Bilodeau-Savaria, and Normand 2022), `exact` (Devriendt 2022), `max-cdcl`, `w-max-cdcl` (Coll et al. 2022a), `max-hs` (Bacchus 2022), `open-wbo` (Martins, Manquinho, and Lynce 2014), `uwr-max-sat-scip`, `uwr-max-sat` (Piotrów 2020), `w-max-cdcl-band-all` (Coll et al. 2022b).

Results

We conducted a comparison on *generated tests*. Environment details can be found in the full version. As illustrated in Fig. 1 the results clearly demonstrate the superior performance of our algorithm over the competing solvers. To carry out the comparison, we set the parameters a and k to fixed values and linearly increased the parameter b. Demonstrating running time on practice of our algorithm matches theoretical upper bound provided for the algorithm.

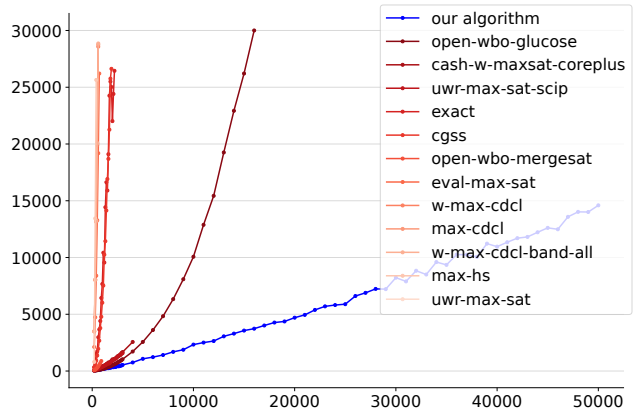

Figure 1: Computational time of various MaxSAT solvers on *generated instances* with $100 \leq b \leq 50000, a = 20, k = 10$. The absence of the point on the graph means that the corresponding algorithm spent more than 30s for the certain instance. Y-axis in milliseconds

For majority of instances from MaxSAT Evaluations 2022 application of our solver is problematic. As usually for these instances value of the parameter is very large. Besides, we do not implement any heuristics that sometimes significantly speed up computations in practice even though they do not have any proven guarantee.

Conclusion

Our findings lead to improvement of algorithms for the following problems $(\nu(F) + k)$-SAT, $(n - k)$-Set Cover. The new running time is $\mathcal{O}^*(2^{\frac{3k}{2}})$. We also show that the same upper bound holds for Partial $(\nu(F) + k)$-SAT if all hard clauses have length at most 2. If this is not the case then we can solve Partial $(\nu(F) + k)$-SAT in time $\mathcal{O}^*(2^{\frac{3k}{2}} + 2^k \cdot 1.12226^h)$ where h is the number of hard clauses in the input formula. Moreover, we establish record upper bounds for $(n, 3)$-MaxSAT and $(n, 4)$-MaxSAT if we measure the running times in terms of the overall number of clauses or in terms of the number of clauses that we need to satisfy. Note that recent algorithms for MaxSAT in terms of the number of clauses that needs to be satisfied reduce problem to the case when only $(4, 1)$, $(3, 1)$ variables appear in the formula. Here, we showed that instead we can try to reduce the problem to an instance where each variable appears at most 4 times, as for such subproblem we have an algorithm.

Moreover, we implemented our algorithm and tested it versus state-of-the-art MaxSAT solvers. On instances for which our algorithm was designed it significantly outperforms competitors. Hence, including ideas of our algorithm as subroutines or subprocedures in modern solvers might further extend area of their applicability.

Acknowledgements

Research is partially supported by Huawei (grant TC20231108096).

Research of Ivan Bliznets is supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 853234).

References

Alferov, V.; and Bliznets, I. 2021. New Length Dependent Algorithm for Maximum Satisfiability Problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 3634–3641.

Avellaneda, F.; Bilodeau-Savaria, C.-E.; and Normand, L. 2022. Weighted version of EvalMaxSAT 2022. In *MaxSAT Evaluation*, volume 16, 12.

Bacchus, F. 2022. MaxHS in the 2022 MaxSat Evaluation. In *MaxSAT Evaluation*, volume 16, 17–18.

Bansal, N.; and Raman, V. 1999. Upper bounds for MaxSat: Further improved. In *International symposium on algorithms and computation*, 247–258. Springer.

Basavaraju, M.; Francis, M. C.; Ramanujan, M.; and Saurabh, S. 2016. Partially polynomial kernels for set cover and test cover. *SIAM Journal on Discrete Mathematics*, 30(3): 1401–1423.

Belova, T.; and Bliznets, I. 2020. Algorithms for (n, 3)-MAXSAT and parameterization above the all-true assignment. *Theoretical Computer Science*, 803: 222–233.

Biere, A.; Heule, M.; and van Maaren, H. 2021. *Handbook of satisfiability*. IOS press.

Brilliantov, K.; Alferov, V.; and Bliznets, I. 2023. Improved Algorithms for Maximum Satisfiability and Its Special Cases. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 3898–3905.

Chen, J.; and Kanj, I. A. 2004. Improved exact algorithms for Max-Sat. *Discrete Applied Mathematics*, 142(1-3): 17–27.

Chen, J.; Xu, C.; and Wang, J. 2015. Dealing with 4-variables by resolution: an improved MaxSAT algorithm. In *Workshop on Algorithms and Data Structures*, 178–188. Springer.

Chen, J.; Xu, C.; and Wang, J. 2017. Dealing with 4-variables by resolution: an improved MaxSAT algorithm. *Theoretical Computer Science*, 670: 33–44.

Coll, J.; Li, S.; Li, C.-M.; Manya, F.; Habet, D.; and He, K. 2022a. MaxCDCL and WMaxCDCL in MaxSAT Evaluation 2022. In *MaxSAT Evaluation*, volume 16, 15–16.

Coll, J.; Li, S.; Li, C.-M.; Manya, F.; Habet, D.; and He, K. 2022b. WMaxCDCL-BandAll in MaxSAT Evaluation 2022. In *MaxSAT Evaluation*, volume 16.

Crowston, R.; Gutin, G.; Jones, M.; Raman, V.; Saurabh, S.; and Yeo, A. 2014. Fixed-parameter tractability of satisfying beyond the number of variables. *Algorithmica*, 68(3): 739–757.

Cygan, M.; Fomin, F. V.; Kowalik, Ł.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized algorithms*, volume 5. Springer.

Devriendt, J. 2022. Exact: evaluating a pseudo-Boolean solver on MaxSAT problems. In *MaxSAT Evaluation*, volume 16, 13–14.

Fomin, F. V.; and Kratsch, D. 2010. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer. ISBN 978-3-642-16532-0.

Gutin, G.; and Mnich, M. 2022. A survey on graph problems parameterized above and below guaranteed values. *arXiv preprint arXiv:2207.12278*.

Hopcroft, J. E.; and Karp, R. M. 1973. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4): 225–231.

Ihalainen, H.; Berg, J.; and J"arvisalo, M. 2021. Refined Core Relaxation for Core-Guided MaxSAT Solving. In *CP*, volume 210 of *LIPIcs*, 28:1–28:19. Schloss Dagstuhl - Leibniz-Zentrum f"ur Informatik.

Lei, Z.; Wang, Y.; Pan, S.; Cai, S.; and Yin, M. 2022. CASHWMaxSAT-CorePlus: Solver Description. In *MaxSAT Evaluation*, volume 16, 8.

Mahajan, M.; and Raman, V. 1999. Parameterizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms*, 31(2): 335–354.

Marek, V. W. 2009. *Introduction to mathematics of satisfiability*. CRC Press.

Martins, R.; Manquinho, V.; and Lynce, I. 2014. Open-WBO: A Modular MaxSAT Solver,. In Sinz, C.; and Egly, U., eds., *Theory and Applications of Satisfiability Testing – SAT 2014*, 438–445. Cham: Springer International Publishing. ISBN 978-3-319-09284-3.

Piotrów, M. 2020. UWrMaxSat: Efficient Solver for MaxSAT and Pseudo-Boolean Problems. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, 132–136. IEEE.

Szeider, S. 2004. Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. *Journal of Computer and System Sciences*, 69(4): 656–674.

Xiao, M. 2022. An Exact MaxSAT Algorithm: Further Observations and Further Improvements. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 1887–1893. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Xu, C.; Li, W.; Yang, Y.; Chen, J.; and Wang, J. 2019. Resolution and domination: an improved exact MaxSAT algorithm. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 1191–1197. AAAI Press.