Approximating MAX SAT by moderately exponential algorithms

Bruno Escoffier, Vangelis Paschos, Emeric Tourniaire

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision

CNRS UMR 7243

CAHIER DU LAMSADE
304

Mars 2011

Approximating MAX SAT by moderately exponential algorithms

Bruno Escoffier, Vangelis Th. Paschos, Emeric Tourniaire

Approximating MAX SAT by moderately exponential algorithms[*]

Bruno Escoffier[1] Vangelis Th. Paschos[1,2] Emeric Tourniaire[1]

[1]LAMSADE, CNRS and Université Paris-Dauphine, France

{escoffier,paschos,tourniaire}@lamsade.dauphine.fr

[2]Institut Universitaire de France

March 16, 2011

Abstract

We study approximation of the MAX SAT problem by moderately exponential algorithms. The general goal of the issue of moderately exponential approximation is to catch-up on polynomial inapproximability, by providing algorithms achieving, with worst-case running times importantly smaller than those needed for exact computation, approximation ratios unachievable in polynomial time. We develop several approximation techniques that can be applied to MAX SAT in order to get approximation ratios arbitrarily close to 1.

1 Introduction

Optimum satisfiability problems are of great interest from both theoretical and practical points of view. On the one hand, SAT is the first complete problem for NP and MAX SAT, MIN SAT have generalizations or restrictions that have been among the first problems proved complete for numerous approximability classes under various approximability preserving reductions [2, 3, 18, 20]. On the other hand, in many fields (including artificial intelligence, database system, mathematical logic, . . .) several problems can be expressed in terms of versions of SAT [4].

Satisfiability problems have in particular drawn a major attention in the field of polynomial time approximation as well as in the field of parameterized and exact solution by exponential time algorithms (see Section 2). Our goal in this paper is to develop approximation algorithms for MAX SAT with running times which, though being exponential, are much lower than those of exact algorithms, and with a better approximation ratio than the one achieved in polynomial time. This approach has already been considered for MAX SAT in [16], where interesting tradeoffs between running time and approximation ratio are given. It has also been considered for several other well known problems such as MINIMUM SET COVER [10, 14], MIN COLORING [6, 9], MAX INDEPENDENT SET and MIN VERTEX COVER [8], MIN BANDWIDTH [15, 19], . . . Similar issues arise in the field of FPT algorithms, where approximation notions have been introduced, for instance, in [11, 17]. In this article, we propose several improvements of the result of [16] using various algorithmic techniques.

[*]Research partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

The paper is organized as follows. In Section 2, we formally define the problem, mention some known results and briefly sketch the ideas leading to some of the results in [16]. We propose first improvements in Section 3: the first one uses the same technique as in [16] while the second one uses a different approach consisting of splitting the instance in "small" sub-instances. In Section 4, we further improve these results for some ratios using another technique consisting of approximately pruning a search tree. All these results deal with complexity depending on the number of clauses. In Section 5, we obtain the same kind of results when we are interested in running times that depend on the number of variables. We conclude the article in Section 6 where we briefly mention the MIN SAT problem.

2 Preliminaries and notations

Given a set of variables and a set of disjunctive clauses, MAX SAT consists of finding a truth assignment for the variables that maximizes the number of satisfied clauses. In what follows, we denote by $X = \{x_1, x_2, \ldots, x_n\}$ the set of variables and by $C = \{C_1, C_2, \ldots C_m\}$ the set of clauses. Each clause consists of a disjunction of literals that are either a variable x_i or the negation of a variable $\neg x_i$. A ρ-approximation algorithm for MAX SAT (with $\rho < 1$) is an algorithm that finds an assignment satisfying at least a fraction ρ of the maximal number of simultaneously satisfied clauses. The best known ratio guaranteed by a polynomial time approximation algorithm is $\alpha = 0.785$ obtained in [1].

Dealing with exact solution, [12] gives an exact algorithm working in time $O^*(1.3247^m)$, which is the best known bound so far. Dealing with the number of variables, the trivial $O^*(2^n)$ bound has not yet been broken down, and this constitutes one of the main open problems in the field of exact exponential algorithms. The parameterized version of MAX SAT consists, given a set of clauses C and an integer k, of finding a truth assignment that satisfies at least k clauses, or to output an error if no such assignment exists. In [12] the authors give a parameterized algorithms for MAX SAT running in time $O^*(1.3695^k)$.

Using the same notation as in [12], we say that that a variable x is an (i, j)-variable if it occurs positive in exactly i clauses and negative in exactly j clauses. For any instance C of MAX SAT, we will denote by $\text{OPT}(C)$ (or OPT if no ambiguity occurs) an optimal set of satisfied clauses. We use notation $f(n) = O^*(g(n))$, if and only if $\exists c \in \mathbb{R}, f(n) = O(g(n)n^c)$. Finally, we denote by α the ratio guaranteed by a polynomial time approximation algorithm (currently $\alpha = 0.785$ is the best value). In general, ρ will denote the approximation ratio of an algorithm, and, when dealing with exponential complexity, γ will be the basis of the exponential expressing it.

In order to fix ideas, let us give a first algorithm, very simple and useful to understand some of our results. In particular, it is one of the basic stones of the results in [16]. It is based upon the following two well known reduction rules.

Reduction 1. Any clause containing an $(n, 0)$- or a $(0, n)$-literal can be removed from the instance. This is correct because we can set this literal to `TRUE` or `FALSE` and satisfy the clauses that contain it.

Reduction 2. Any $(1, 1)$-literal can be removed too. Let $C_1 = x_1 \vee x_2 \vee \cdots \vee x_p$ and $C_2 = \neg x_1 \vee x'_2 \vee \cdots \vee x'_q$ be the only two clauses containing the variable x_1. If there exist two opposite literals in C_1 and C_2, then we can satisfy these clauses by choosing a correct value for x_1 and therefore we can remove these clauses. Otherwise, we can replace these clauses by $C = x_2 \vee \cdots \vee x_p \vee x'_2 \vee \cdots \vee x'_q$. The optimum in the initial instance is the optimum in the reduced instance plus 1.

Algorithm 1. Build a tree as follows. Each node is labeled with a sub-instance of MAX SAT. The root is the initial instance. The empty instances are the leaves. For each node whose label is a non-empty sub-instance, if one of the reductions above applies, then the node has one child labeled with the resulting (reduced) sub-instance. Else, a variable x is arbitrarily chosen and the node has two children: in the first one, the instance has been transformed by setting x to `FALSE` (the literals $\neg x$ have been removed and the clauses containing the literal $\neg x$ are satisfied); in the second one, x is set to `TRUE` and the contrary happens. Finally, for both children the empty clauses are marked unsatisfied. Thus, each node represents a partial truth assignment. An optimal solution is a truth assignment corresponding to a leaf that has the largest number of satisfied clauses. ■

To evaluate the complexity of Algorithm 1, we count the number of leaves in the tree (note that if the number of leaves is $T(n)$, then the algorithm obviously works in time $O^*(T(n))$; in the sequel, in order to simplify notations we will use $T(n)$ to denote both the number of leaves (when we express recurrences) and the complexity). There are two ways to count the number of leaves:

- On the one hand, each node has two children for which the number of remaining variables decreases by 1. This leads to a number of leaves $T(n) \leqslant 2 \times T(n-1)$ and therefore $T(n) = O^*(2^n)$.
- On the other hand, on each node, if the chosen variable is an (i, j)-variable, then the first child will have its number of clauses decreased by at least i and the second child by at least j. The worst case, using the two reduction rules given above, is $i = 1$ and $j = 2$, that leads to $T(m) = T(m-1) + T(m-2)$ and therefore $T(m) = O^*(1.618^m)$.

In [16], the authors showed a way to transform any polynomial time approximation algorithm (with ratio α) into an approximation algorithm with ratio ρ (for any $\alpha \leq \rho \leq 1$) and running time $O^*(1.618^{(\rho-\alpha)(1-\alpha)^{-1}m})$. The basic idea of this algorithm is to build the same tree as in Algorithm 1 up to the fact that we stop the branching when enough clauses are satisfied. Then the α-approximation polynomial algorithm is applied on the resulting sub-instances. As already mentioned, the best value of α is 0.785 [1].

3 First improvements

We provide in this sections two first improvements of the result given in [16]. The first one, given in Section 3.1, uses the same idea as [16] while the second one uses a completely different technique and achieve improved running times (for some approximation ratios) by splitting the initial instance in sub-instances of smaller size.

3.1 Using a better parameterized algorithm

In this section we briefly mention that the same technique as in [16] leads to an improved result when we build the search tree according to the algorithm from [12] instead of the branching tree presented in Section 2. We so derive the following algorithm, that is strictly better than the one of [16] (Figure 1).

Algorithm 2. Build a search-tree as the parameterized algorithm of [12] does. Stop the development of this tree at each node where at least $(m\,(\rho-\alpha)\,/\,(1-\alpha))$ clauses are satisfied (recall that α is the best known polynomial approximation ratio for MAX SAT), or when the instance is empty. For each leaf of the so-pruned tree, apply a polynomial α-approximation algorithm to

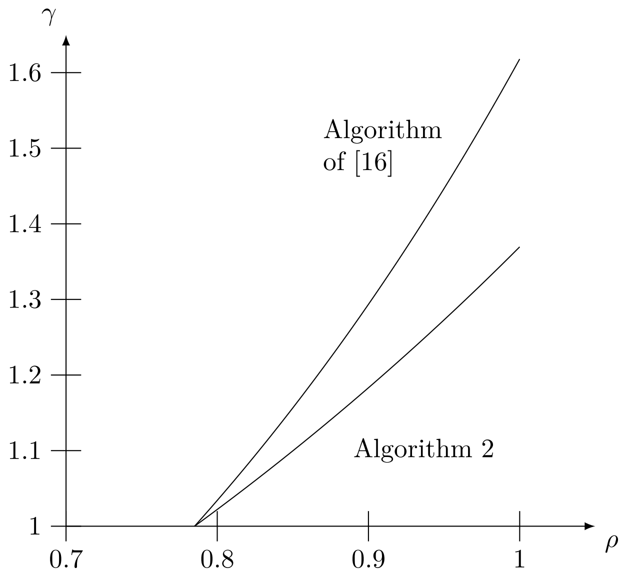


Figure 1: Comparison between the algorithm of [16] and Algorithm 2.

complete the assignment of the remaining variables; thus, each leaf of the tree corresponds to a complete truth assignment. Return the assignment satisfying the largest number of clauses. ■

Theorem 1. *For any ρ such that $\alpha \leq \rho \leq 1$, Algorithm 2 achieves approximation ratio ρ in time $O^*(1.3695^{m(\rho-\alpha)/(1-\alpha)})$.*

Proof. First, let us consider the running time. The parameterized algorithm of [12] builds a search tree where the worst case recurrence relation is $T(k) \leq 2T(k-3) + 2T(k-7)$, where the parameter k is the number of satisfied clauses, leading to a global complexity of $O^*(1.3695^k)$. Here, we build this tree and stop the construction in each leave where $m(\rho-\alpha)/(1-\alpha)$ clauses are satisfied. This leads to a running time of $O^*(1.3695^{m(\rho-\alpha)/(1-\alpha)})$.

No, let us consider the approximation ratio. First, if the number of satisfied clauses $|\mathrm{OPT}|$ by an optimum solution OPT is less than $m(\rho-\alpha)/(1-\alpha)$, then Algorithm 2 obviously finds an optimum solution. Otherwise, let us consider the branch of the branching tree where the leaf corresponds to a partial optimal truth assignment satisfying clauses in OPT. Denote by k_0 the number of clauses satisfied in this leaf ($k_0 \geqslant m(\rho-\alpha)/(1-\alpha)$), i.e. by the partial assignment corresponding to this leaf. Using this assignment, we get a resulting instance in which it is possible to satisfy $|\mathrm{OPT}| - k_0$ clauses (because the optimal assignment satisfies $|\mathrm{OPT}|$ clauses). Consequently, the α-approximation algorithm called by Algorithm 2 will satisfy at least $\alpha\,(|\mathrm{OPT}| - k_0)$ more clauses.

So, finally, at least $k_0 + \alpha\,(|\mathrm{OPT}| - k_0) = k_0(1-\alpha) + \alpha|\mathrm{OPT}| \geqslant m\,(\rho-\alpha) + \alpha|\mathrm{OPT}| \geqslant |\mathrm{OPT}|(\rho-\alpha) + \alpha|\mathrm{OPT}| = \rho|\mathrm{OPT}|$ clauses will be satisfied. □

3.2 Splitting the clauses

In [8, 7], it is shown that a generic method can give interesting moderately exponential approximation algorithms if applied in (maximization) problems satisfying some hereditary property (a

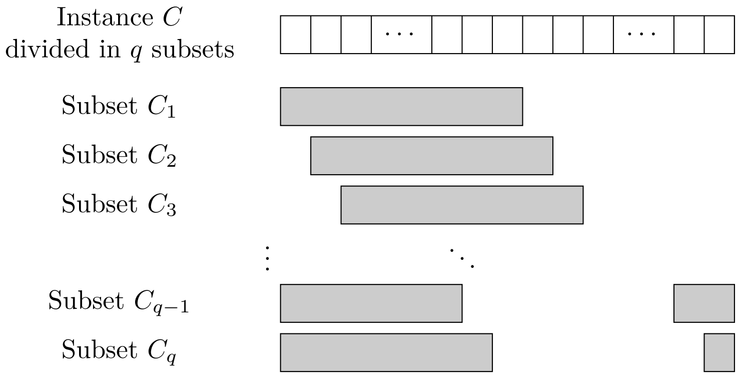


Figure 2: Forming the q subsets of clauses.

property is said to be hereditary if for any set A satisfying this property, and any $B \subset A$, B satisfies this property too).

MAX SAT can be seen as searching for a maximum subset of clauses satisfying the property "can be satisfied by a truth assignment", and this property is clearly hereditary. Therefore, we can adapt the splitting method introduced in [8, 7] to transform any exact algorithm into a ρ-approximation algorithm, for any rational ρ, and with running time exponentially increasing with ρ.

Algorithm 3. Let p, q be two integers such that $\rho = p/q$. Split the set of clauses into q pairwise disjoint subsets $A_1, \cdots, A_q$ of size m/q (at most $\lceil m/q \rceil$ if m/q is not an integer). Then, consider as described in Figure 2 the q subsets $C_i = A_i \cup A_{i+1} \cup \cdots \cup A_{i+p-1}$ (if the index is larger than q, take it modulo q) for $i = 1, \cdots, q$. On each subset, apply some exact algorithm for MAX SAT. Return the best truth assignment among them as solution for the whole instance. ■

Theorem 2. *Given an exact algorithm for* MAX SAT *running in time $O^*(\gamma^m)$, Algorithm 3 achieves approximation ratio ρ in time $O^*(\gamma^{\rho m})$.*

Proof. Algorithm 3 calls q times an exact algorithm (whose running time is $O^*(\gamma^m)$). Then the bound of the running time easily follows from the fact that each subset C_i contains at most $p\lceil m/q \rceil \leq \rho m + p$ clauses.

For the approximation ratio, note first that if we restrict an instance with a set C of clauses to a new instance with a new set $C' \subset C$ of clauses, then an optimal solution for C' satisfies at least the same amount of clauses in C' than an optimal solution for C (in other words, the restriction of any solution for C to C' is feasible for C'), i.e., $|\text{OPT}(C) \cap C'| \leqslant |\text{OPT}(C')|$. In particular, for $i = 1, \ldots, q$, $|\text{OPT}(C_i)| \geqslant |\text{OPT}(C) \cap C_i|$.

Now, note that by construction of the C_i's, we easily see that each clause appears in exactly p among the q subsets $C_1, C_2, \ldots, C_q$, and this holds in particular for any clause in OPT. Thus, $\sum_{i=1}^{q} |\text{OPT}(C) \cap C_i| = p \times |\text{OPT}(C)|$. By the discussion above, $\sum_{i=1}^{q} |\text{OPT}(C_i)| \geqslant p \times |\text{OPT}(C)|$. Since $\sum_{i=1}^{q} |\text{OPT}(C_i)| \leqslant q \times \max_{i=1}^{q} |\text{OPT}(C_i)|$, then $\max_{i=1}^{q} |\text{OPT}(C_i)| \geqslant \frac{p}{q} |\text{OPT}(C)|$. □

It is worth noticing that Algorithm 3 is faster that Algorithm 2 for ratios close to 1 (see Figure 3 in Section 4).

4 Approximate pruning of the search tree

Informally, the idea of an approximate pruning of the search tree is based upon the fact that, if we seek, say, a 1/2-approximation for a maximization problem, then when a search-tree based algorithm selects a particular datum d for inclusion in the solution, one may remove one other datum d' from the instance (without, of course, including it in the solution). At worst, d' is part of an optimal solution and is lost by our solution. Thus, globally, the number of data in an optimum solution is at most two times the number of data in the built solution. On the other hand, with the removal of d', the size of the surviving instance is reduced not by 1 (due to the removal of d) but by 2.

This method can be adapted to MAX SAT in the following way: revisit Algorithm 1 and recall that its worst case with respect to m is to branch on a $(1, 2)$-literal and to fix 1 (satisfied) clause on the one side and 2 (satisfied) clauses on the other side. If we decide to also remove 1 more clause (arbitrarily chosen) in the former case and 2 more clauses (arbitrarily chosen) in the latter one, this leads to a running time $T(m)$ satisfying $T(m) \leqslant T(m-2) + T(m-4)$, i.e., $T(m) \leqslant O^*(1.27^m)$. Since in the branches we have satisfied at least $s \geq 1$ clause (resp., $s \geq 2$ clauses) while the optimum satisfies at most $s+1$ clauses (resp., $s+2$ clauses), we get an approximation ratio 0.5.

This basic principle is not sufficient to get interesting result for MAX SAT, but it can be improved as follows. Let us consider the left branch where the $(1, 2)$-literal is set to true, satisfying a clause C_1. Instead of throwing away one other clause, we pick two clauses C_2 and C_3 such that C_2 contains a literal ℓ and C_3 contains the literal $\neg\ell$, and we remove these two clauses. Any truth assignment satisfies either C_2 or C_3, meaning that in this branch we will satisfy at least 2 clauses (C_1 and one among C_2 and C_3), while at worst the optimum will satisfy these three clauses. In the other branch where 2 clauses are satisfied, we pick two pairs of clauses containing opposite literals and we remove them. This trick improves both the approximation ratio and the running time: now we have an approximation ratio 2/3 (2 clauses satisfied among 3 clauses removed in one branch, 4 clauses satisfied among 6 clauses removed in the other branch), and the running time satisfies $T(m) \leqslant T(m-3) + T(m-6)$, i.e., $T(m) = O^*(1.17^m)$.

In what follows, we generalize the ideas sketched above in order to work for any ratio $\rho \in \mathbb{Q}$.

Algorithm 4. Let p and q be two integers such as $\frac{p}{q} = \frac{\rho-1}{1-2\rho}$. We build the search tree and, on any of its nodes, we count the number of satisfied clauses since the root (we do not count here the clauses that have been arbitrarily removed). Each time we reach a multiple of q, we pick p pairs of clauses with opposite literals and we remove them from the remaining sub-instance. When such a sub-instance on a node is empty, we arbitrarily assign a value on any still unassigned variable. Finally, we return the best truth assignment so constructed. ■

Note that it might be the case that at some point it is impossible to find p pairs of clauses with opposite literals. But this means that (after removing $q < p$ pairs) each variable appears only positively or only negatively, and the remaining instance is clearly easily solvable in linear time.

Theorem 3. *Algorithm 4 satisfies at least $\rho|OPT|$ clauses and runs in time $O^*(1.618^{m(2\rho-1)})$.*

Proof. Consider the leaf where the variables are set like in an optimum solution. In this leaf, assume that the number of satisfied clauses is $s \times q + s'$ (where $s' < q$) - again, we do not count the clauses that have been arbitrarily removed. Then, the algorithm has removed $s \times 2p$ clauses arbitrarily, among which at least $s \times p$ are necessarily satisfied. In the worst case, the $s \times p$ other clauses are in OPT; hence, $|\text{OPT}| \leqslant 2sp + sq + s'$. So, the approximation ratio of Algorithm 4 is at least:

$$\frac{sq+sp+s'}{sq+2sp+s'} \geq \frac{q+p}{q+2p} = \frac{1+\frac{p}{q}}{1+2\frac{p}{q}} = \rho$$

We now estimate the running time of Algorithm 4. For each node i of the tree, denote by m_i the number of clauses left in the surviving sub-instance of this node, by z_i the number of satisfied clauses since the root of the tree (we do not count the clauses that have been arbitrarily removed) and set $t_i = m_i - (2p/q)(z_i \mod q)$.

For the root of the tree, $z_i = 0$ and therefore $t_i = m$. Let i be a node with two children j (at least one clause satisfied) and g (at least two clauses satisfied). Let us examine quantity t_j when exactly one clause is satisfied. In this case, $z_j = z_i + 1$. On the other hand:

i) If $z_j \mod q \neq 0$, then we do not have reached the threshold necessary to remove the $2p$ clauses. Then, $m_j = m_i - 1$ and $t_j = m_j - 2p/q(z_j \mod q) = m_i - 1 - 2p/q((z_i \mod q)+1) = t_i - 1 - 2p/q$.

ii) If $z_j \mod q = 0$, then $z_i \mod q = q-1$ and the threshold has been reached; so $2p$ clauses have been removed. Then, $m_j = m_i - 1 - 2p$, $t_j = m_j = m_i - 1 - 2p$ and $t_i = m_i - 2p/q(q-1) = m_i - 2p + 2p/q$. Finally, $t_j = t_i - 1 - 2p/q$.

Therefore, in both cases, $t_j \leqslant t_i - 1 - 2p/q$. Of course, by a similar argument, if we satisfy g clauses, then the quantity t_i is reduced by $g(1+2p/q)$. This leads to a running time $T(t) \leqslant T(t-1-2p/q) + T(t-2-4p/q)$ and hence $T(t) = 1.618^{t/(1+2p/q)}$. Since initially $t = m$, we get $T(m) = 1.618^{m/(1+2p/q)}$. Taking into account that $p/q = (\rho-1)/(1-2\rho)$, we get immediately $1/(1+2p/q) = 2\rho - 1$, that completes the proof. □

Finally, Algorithm 4 can be improved if instead of using the simple branching rule in the tree, the more involved case analysis of [12] is used. This derives the following algorithm.

Algorithm 5. Let p and q be two integers such as $\frac{p}{q} = \frac{\rho-1}{1-2\rho}$. Build the search-tree of [12] and, on each node of it, count the number of satisfied clauses since the root. Each time a multiple of q is reached, pick p pairs of clauses with opposite literals and remove them from the resulting sub-instance. Return the best truth assignment so constructed. ■

To estimate the running time of Algorithm 5, we use nearly the same analysis as in [12]. The only difference is that, at each step of the branching tree, [12] counts without distinction the satisfied and the unsatisfied clauses (because left empty), whereas we have to make a difference in the complexity analysis: a satisfied clause reduces the quantity t by $1+2p/q$ in our algorithm, while an unsatisfied clause reduces it by only 1.

We illustrate this analysis for the case 4.2 of [12] ("there is $(2,2)$-literal x that occurs at least once as a unit clause"). In this case, a branching is done on the variable x. On the one side, 2 clauses are satisfied while, on the other side, 2 are satisfied and 1 becomes empty and thus it is removed from the instance. For [12], this leads to a complexity $T(m) \leqslant T(m-2) + T(m-3)$. For Algorithm 5, this gives $T(m) \leqslant T(m-2-4p/q) + T(m-3-4p/q)$. To simplify these results, set $\chi = 2p/q$. Then $T(m) \leqslant T(m-2-2\chi) + T(m-3-2\chi)$, which leads to $T(m) = O^*(\gamma^m)$ with γ the largest real solution of the equation $\gamma^{2\chi+3} - \gamma - 1 = 0$.

For the other cases, a comparative study between the algorithm of [12] and Algorithm 5 is summarized in Table 1. Its third column gives equations whose largest real solutions are the worst case running times for Algorithm 5.

Now, depending on the seeked ratio ρ, the running time of the algorithm is given by the worst case of all the cases given in Table 1. However, one can show that for any ρ the worst case is always reached by the case 4.2. Let us show an example (the other cases are similar). Consider

Case	[12]	Algorithm 5
4.0 a)	$T(m) = T(m-1) + T(m-5)$	$T(m) = T(m-1-\chi) + T(m-5-4\chi)$
4.0 b)	$T(m) = T(m-1) + T(m-7) + T(m-10)$	$T(m) = T(m-1-\chi) + T(m-7-6\chi) + T(m-10-9\chi)$
4.1	$T(m) = 2T(m-3)$	$T(m) = 2T(m-3-3\chi)$
4.2	$T(m) = T(m-2) + T(m-3)$	$T(m) = T(m-2-2\chi) + T(m-3-2\chi)$
4.3	$T(m) = 2T(m-6) + T(m-2)$	$T(m) = 2T(m-6-6\chi) + T(m-2-2\chi)$
4.4	$T(m) = T(m-3) + T(m-2)$	$T(m) = T(m-3-3\chi) + T(m-2-2\chi)$
4.5	Same as for 4.3	
4.6	Same as for 4.4	
4.7	$T(m) = 2T(m-5)$	$T(m) = 2T(m-5-5\chi)$
4.8	$T(m) = 2T(m-5) + 2T(m-7)$	$T(m) = 2T(m-5-5\chi) + 2T(m-7-6\chi)$
4.9 a)	Same as for 4.1	
4.9 b)	Same as for 4.0 a)	
4.10	$T(m) = T(m-1) + 1$	$T(m) = T(m-1) + 1$
4.11 a)	$T(m) = 2T(m-4)$	$T(m) = 2T(m-4-4\chi)$
4.11 b)	Same as for 4.0 a)	
4.12 a)	Same as for 4.0 a)	
4.12 b)	$T(m) = 2T(m-8) + T(m-1)$	$T(m) = 2T(m-8-7\chi) + T(m-1-\chi)$

Table 1: Running times for the algorithm of [12] and Algorithm 5.

the equations $f_{4.2}(X) = X^{2\chi+3} - X - 1 = 0$ and $f_{4.0}(X) = X^{4\chi+5} - X^{3\chi+4} - 1 = 0$. The largest real solution of the former is always larger than the largest real solution of the latter one. Indeed, let χ be any positive value. Remark first that $f'_{4.0}(X) = (4\chi+5)X^{4\chi+4} - (3\chi+4)X^{3\chi+3} > 0$; hence, function $f_{4.0}$ is increasing with $X > 1$. What we now need to show is that if $f_{4.2}(X) = 0$, then $f_{4.0}(X) \geqslant 0$ (this means that the zero of $f_{4.0}$ is before that of $f_{4.2}$):

$$\begin{aligned}
f_{4.2}(X) = 0 \quad &\Leftrightarrow \quad X^{2\chi+3} = X + 1 \quad \Leftrightarrow \quad X^{3\chi+4} = X^{\chi+2} + X^{\chi+1} \\
&\Leftrightarrow \quad X^{4\chi+5} = X^{2\chi+3} + X^{2\chi+2} \quad \Leftrightarrow \quad X^{4\chi+5} = X^{2\chi+2} + X + 1 \\
&\Rightarrow \quad X^{4\chi+5} - X^{3\chi+4} - 1 = X^{2\chi+2} + X - X^{\chi+2} - X^{\chi+1} \\
&\Rightarrow \quad f_{4.0}(X) = X^{2\chi+2} + X - X^{\chi+2} - X^{\chi+1} \\
&\Rightarrow \quad f_{4.0}(X) = \left(X^{\chi+1} - 1\right)\left(X^{\chi+1} - X\right) \geqslant 0
\end{aligned}$$

and the result follows.

Also, it is easy to see that the result of Theorem 3 dealing with the approximation ratio of Algorithm 4 identically applies also for Algorithm 5. Putting all the above together, the following theorem holds and concludes the section.

Theorem 4. *For any $\rho < 1$, Algorithm 5 achieves approximation ratio ρ on* MAX SAT *with running time $T(m) = O^*(\gamma^m)$, where γ is real solution of the equation $X^{2\alpha+3} - X - 1 = 0$ and $\alpha = \frac{2\rho-2}{1-2\rho}$.*

Figure 3 illustrates the relationship approximation ratio - running time of the different methods seen so far. The numeric analysis shows that, with the current value of α, and with the algorithm of [12] as the best currently known algorithm for MAX SAT, Algorithm 2 is the most efficient for ratios less than 0.967 while Algorithm 5 dominates the other ones for ratios above this value.

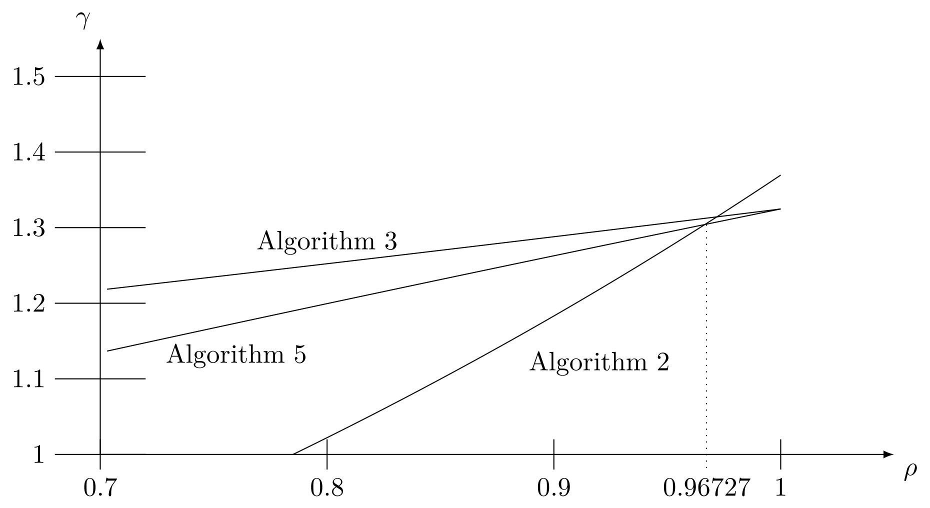


Figure 3: Evaluation of the running times for the different methods.

5 Splitting the variables

In this section, we present two algorithms that approximate MAX SAT within any approximation ratio smaller than 1, and with a computation time depending on n (the number of variables). The first algorithm of this section builds several trees. Then, in each of them, as for Algorithm 2 in Section 3.1, it cuts the tree at some point and completes variables' assignment using a polynomial approximation algorithm, as it is illustrated in Figure 4.

Algorithm 6. Let p and q be two integers such that $p/q = (\rho - \alpha)/(1 - \alpha)$. Build q subsets $X_1, \cdots, X_q$ of variables, each one containing roughly $p/q \times n$ variables, where each variable appears in exactly p subsets (as in Figure 2 in Section 3.2). For each subset X_i, construct a complete branching tree, considering only the variables in the subset (i.e., the depth of each of these trees is exactly $|X_i| \simeq p/q \times n$). For each of the leaves of these trees, run a polynomial time algorithm guaranteeing a ratio α on the surviving sub-instance. Return the best truth assignment among those built. ■

Each of the trees built by Algorithm 6 is a binary tree and has depth roughly pn/q (at most $pn/q + p$ to be precise). So its running time is $O^*(2^{np/q})$. Note also that, on each of these trees, at least one leaf is a partial assignment of an optimal (global) truth assignment. We will call such a leaf an *optimal leaf*.

Lemma 1. *At least one of optimal leaf has at least $\frac{p}{q} \times |OPT|$ satisfied clauses (before applying the polytime approximation algorithm).*

Proof. Remark that every clause C_i in OPT contains at least one true literal; pick one of them from each clause C_i and denote the variable corresponding to this literal by $\mathrm{Var}(C_i)$. Let, for each variable x, $C(x)$ be the set of clauses from OPT for which x or $\neg x$ is the picked literal, i.e., $\forall x \in X$, $C(x) = \{C_i \in \mathrm{OPT}/\mathrm{Var}(C_i) = x\}$. Based upon this, $\mathrm{OPT} = \bigcup_{x \in X} C(x)$.

In the tree obtained on the set X_i, denote by Λ_i the set of satisfied clauses on some optimal leaf and set $\lambda_i = |\Lambda_i|$. Then, $\bigcup_{x \in X_i} C(x) \subseteq \Lambda_i$ and, by construction, $\forall i, j$, $C(x_i) \cap C(x_j) = \emptyset$.

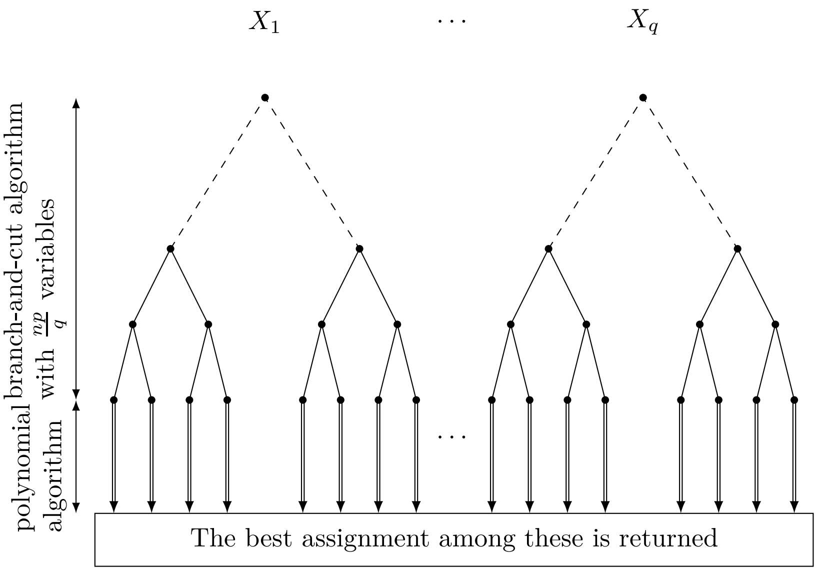

X_1
$\cdots$
X_q
branch-and-cut algorithm with $\frac{np}{q}$ variables
polynomial algorithm
The best assignment among these is returned

Figure 4: Illustration of Algorithm 6.

We so have:

$$\begin{aligned}\lambda_i &\geqslant \sum_{x\in X_i} |C(x)| \\ \sum_{i=1}^{q} \lambda_i &\geqslant \sum_{i=1}^{q}\sum_{x\in X_i} |C(x)|\end{aligned}$$

As every x belongs to exactly p subsets among the q sets X_i, it holds that:

$$(1)\quad \sum_{i=1}^{q} |\lambda_i| \geqslant p \times \sum_{x\in X} |C(x)| = p \times |\mathrm{OPT}|$$

From (1), it is immediately derived that:

$$\max_{i=1}^{q} |\lambda_i| \geqslant \frac{1}{q}\sum_{i=1}^{q} |\lambda_i| \geqslant \frac{p}{q} \times \sum_{x\in X} |C(x)| = \frac{p}{q} \times |\mathrm{OPT}|$$

that concludes the proof. □

Proposition 1. *Algorithm 6 achieves approximation ratio ρ.*

Proof. By Lemma 1, among all the optimal leaves, at least one satisfies $\lambda \geqslant \frac{p}{q} \times |\mathrm{OPT}|$ clauses. As an optimal leaf corresponds to an optimal truth assignment, it is possible to complete this assignment into an optimal (global) solution. In other words, there exist $|\mathrm{OPT}| - \lambda$ remaining clauses that become true on the surviving sub-instance. If the polynomial algorithm called by Algorithm 6 achieves approximation ratio α, it will compute a solution that satisfies at least $\alpha \times (|\mathrm{OPT} - |\lambda|)$ clauses. Hence, the number of satisfied clauses will be at least:

$$|\lambda| + \alpha \times (|\mathrm{OPT} - |\lambda|) = \alpha|\mathrm{OPT}| + (1-\alpha)\lambda \geqslant \alpha|\mathrm{OPT}| + (1-\alpha)\frac{p}{q}|\mathrm{OPT}|$$

that leads to an approximation ratio of $\alpha + (1-\alpha)\frac{p}{q} = \rho$. □

Putting all the above together, the following theorem holds.

Theorem 5. *Algorithm 6 achieves approximation ratio ρ in time $O^*(2^{n(\rho-\alpha)/(1-\alpha)})$, for any $\rho \leq 1$.*

The previous algorithm builds a full branching tree on each subset of variables. In particular, when the seek ratio ρ tends to 1, the basis of the exponent in the complexity tends to 2. Then, one might ask the following question: suppose that there is an exact algorithm solving MAX SAT in $O^*(\gamma^n)$ (for some $\gamma < 2$), is it possible to find a ρ approximation algorithm in time $O^*(\gamma_\rho^n)$ where $\gamma_\rho < \gamma$ for some $\rho \in]\alpha, 1]$? for any $\rho \in]\alpha, 1]$? This kind of reduction from an approximate solution to an exact one would allow to take advantage of any possible improvement of the exact solution of MAX SAT, which is not the case in Algorithm 6. Note that finding an exact algorithm in time $O^*(\gamma^n)$ for some $\gamma < 2$ is a famous open question for MAX SAT as well as for some other combinatorial problems. It has very recently received a positive answer for the Hamiltonian cycle problem in [5].

We propose in the following a positive answer to our question, i.e., ρ-approximation algorithms working in time $O^*(\gamma_\rho^n)$ with $\gamma_\rho < \gamma$ for any $\rho \in]\alpha, 1]$. We first give a simple solution (Algorithm 7) that we improve later (Algorithm 8). Algorithm 7 moves in the same spirit as Algorithm 6 but, instead of building a full branching tree on X_i, calls an exact algorithm on the sub-instance induced by the set X_i.

Algorithm 7. Let $\rho \in \mathbb{Q}$ and p and q two integers such that $p/q = \rho$. Build q subsets of variables, each one containing $p/q \times n$ variables (as in Algorithm 6). For each subset of variables X_i:

a) Remove from the instance the variables not in X_i and any empty clause.

b) Run the exact algorithm on the resulting sub-instance, thus obtaining a truth assignment for the variables in X_i.

c) Complete this assignment with arbitrary truth-values for the variables not in X_i.

Among all the truth assignments produced, return the one that satisfies the largest number of clauses in the whole instance. ■

In Algorithm 7, the exact algorithm called in step b) runs in time $O^*(\gamma^{\rho n})$. Its approximation ratio is the one claimed in Lemma 1. Indeed, the exact algorithm satisfies at least the same amount of clauses as the optimal branching (for the global instance) would do. More precisely, for each X_i, and for any $x \in X_i$, the clauses containing x (and in particular the clauses in $C(x)$) are not removed from the instance. The optimal branching would then satisfy at least $\sum_{x \in X_i} |C(x)|$ clauses and, obviously, the exact algorithm would satisfy even more. In other words, we have the following results.

Proposition 2. *Algorithm 7 achieves approximation ratio ρ in time $O^*(\gamma^{\rho n})$, where $O^*(\gamma^n)$ is the running time of an exact algorithm for MAX SAT.*

As one can see, in step c of Algorithm 7, variables outside X_i, $i = 1, \ldots, q$, are assigned arbitrarily, so, at worst their truth value may satisfy no additional clause. Note that one might want to use an approximation algorithm in the remaining instance as in Algorithm 6; however, the same analysis would not work since the exact solution obtained by the exact algorithm on the sub-instance might be completely different from the partial assignment of a global optimal solution. Nevertheless, we are able to propose an improvement by completing partial solutions in such a way that, roughly speaking, at least half of the remaining clauses are satisfied.

Algorithm 8. Let $p, q \in \mathbb{Q}$ be such that $p/q = 2\rho - 1$. Build q subsets of variables $X_1, \dots, X_q$, each one containing $p/q \times n$ variables (as in Algorithm 7). For each X_i proceed as follows:

i) Assign weight 2 to every clause.

ii) Remove from the instance the variables not in X_i. For each clause missing (at least) one variable from X_i set its weight to 1; remove empty clauses.

iii) Solve exactly this max weighted Sat resulting instance, thus obtaining a truth assignment for the variables in X_i.

iv) Complete the assignment with a greedy algorithm: for each (i, j)-literal, if $i > j$, then the literal is set to **TRUE**, else it is set to **FALSE** (and the instance is modified accordingly).

Return the best among the truth-assignments so-produced. ■

Lemma 2. *If there is a* MAX SAT*-algorithm working in time $O^*(\gamma^n)$, then the instances of max weighted Sat in Algorithm 8 can be solved with the same bound on the running time.*

Proof. Note that the only weights assigned by Algorithm 8 are 1 and 2. In such a weighted instance, we can add a new variable x_0 and replace each clause c of weight 2 by three new clauses: c, $c \vee x_0$ and $c \vee \neg x_0$. Thus, if c is satisfied, then it will count in the new instance as three satisfied clauses. Otherwise, exactly one of the three new clauses will be satisfied. Thus, the so-built instance of MAX SAT is equivalent to the initial MAX WEIGHTED SAT-instance built by Algorithm 7. □

Theorem 6. *Algorithm 8 achieves approximation ratio ρ in time $O^*(\gamma^{(2\rho-1)n})$, where $O^*(\gamma^n)$ is the running time of an exact algorithm for* MAX SAT.

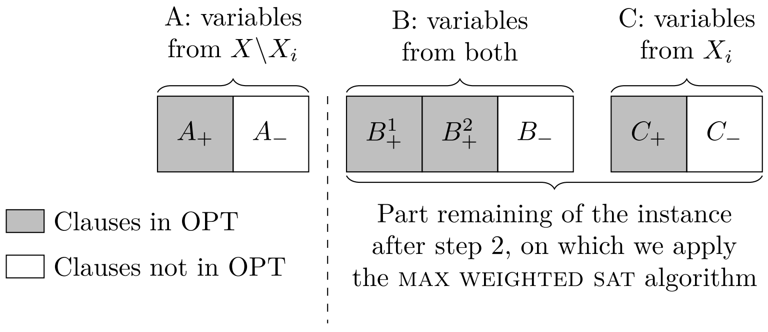
Proof. For the running time: we apply q times an exact algorithm $O^*(\gamma^n)$ on instances of size $(2\rho - 1)n$.

For the approximation ratio, using the same notation as before, consider one particular literal in each clause satisfied by some optimum solution OPT, and let $C(x)$ be the subset of these clauses such that the picked literal is x or $\neg x$. Then, as shown before, there exists a subset X_i such that $\sum_{x \in X_i} |C(x)| \geqslant \frac{p}{q}|\text{OPT}|$. Then, let us consider such a X_i, and denote by (see Figure 5):

- A the subset of clauses containing only variables in $X \setminus X_i$, A_+ (resp., A_-) the subset of clauses from A that are in the optimum (resp., are not in the optimum).
- B the subset of clauses containing at least one variable in X_i and one variable in $X \setminus X_i$; B_+^1 (resp., B_+^2) the subset of clauses from B that are in the optimum and whose chosen variable $\text{Var}(c)$ is in X_i (resp., not in X_i); B_- the subset of clauses from B that are not in the optimum.
- C the remaining clauses, i.e., the clauses that contain only variables in X_i, C_+ (resp., C_-) the subset of clauses from C that are in the optimum (resp., are not in the optimum).

Note that when removing variables in $X \setminus X_i$, clauses in A become empty, so the remaining clauses are exactly those in $B \cup C$. With these notations, $\text{OPT} = A_+ \cup B_+^1 \cup B_+^2 \cup C_+$ and $\sum_{x \in X_i} |C(x)| = |B_+^1| + |C_+|$. Then, for the chosen X_i:

$$(2) \quad |B_+^1| + |C_+| \geqslant \frac{p}{q}|\text{OPT}| = \frac{p}{q}\left(|A_+| + |B_+^1| + |B_+^2| + |C_+|\right)$$



A: variables from $X\backslash X_i$
B: variables from both
C: variables from X_i
A_+
A_-
B^1_+
B^2_+
B_-
C_+
C_-
Clauses in OPT
Clauses not in OPT
Part remaining of the instance after step 2, on which we apply the MAX WEIGHTED SAT algorithm

Figure 5: Division of clauses according to a subset X_i of variables.

With respect to step iii) of Algorithm 8, denote by B_1 the subset of satisfied clauses from B and by C_1 the subset of satisfied clauses from C. As OPT is a particular solution of weight $|B^1_+| + 2|C_+|$ for this weighted Sat problem, we have:

(3) $\quad |B_1| + 2\,|C_1| \geqslant \left|B^1_+\right| + 2\,|C_+|$

The greedy algorithm in step iv) will satisfy at least half of the remaining clauses containing at least one literal from $X \setminus X_i$, i.e., the set $(B\backslash B_1) \cup A$. Finally, the number of satisfied clauses is at least:

$$\begin{aligned} |B_1| + |C_1| + \frac{|B| - |B_1| + |A|}{2} &= |C_1| + \frac{|B_1|}{2} + \frac{|B|}{2} + \frac{|A|}{2} \overset{(3)}{\geqslant} \frac{\left|B^1_+\right|}{2} + |C_+| + \frac{|B|}{2} + \frac{|A|}{2} \\ &\geqslant \left|B^1_+\right| + |C_+| + \frac{\left|B^+_2\right|}{2} + \frac{|A^+|}{2} \end{aligned}$$

So, the approximation ratio achieved is at least:

$$\frac{\left|B^1_+\right| + |C_+| + \frac{\left|B^2_+\right| + |A_+|}{2}}{\left|B^1_+\right| + |C_+| + \left|B^2_+\right| + |A_+|} = \frac{1}{2}\left(1 + \frac{\left|B^1_+\right| + |C_+|}{\left|B^1_+\right| + |C_+| + \left|B^2_+\right| + |A_+|}\right) \overset{(2)}{\geqslant} \frac{1}{2}\left(1 + \frac{p}{q}\right) = \frac{q+p}{2q} = \rho$$

that completes the proof. □

For instance, suppose that MAX SAT is solvable in $O^*(1.657^n)$, which is the running time to solve Hamiltonian cycle in [5]. Then Algorithm 8 achieves a 0.9-approximation in time $O^*(1.576^n)$ while Algorithm 6 achieves the same ratio in time $O^*(1.703^n)$.

6 Discussion

We have proposed in this paper several algorithms that constitute a kind of "moderately exponential approximation schemata" for MAX SAT. They guarantee approximation ratios that are unachievable in polynomial time unless $\mathbf{P} = \mathbf{NP}$. To obtain these schemata, several techniques have been used coming either form the polynomial approximation or from the exact computation. Furthermore, Algorithm 8 in Section 5 is a kind of polynomial reduction between exact computation and moderately exponential approximation transforming exact algorithms running on "small" sub-instances into approximation algorithms guaranteeing good ratios for the whole instance. We think that research in moderately exponential approximation is an interesting research issue for overcoming limits posed to the polynomial approximation due to the strong inapproximability results proved in the latter paradigm.

We conclude this paper with a word about another very well known optimum satisfiability problem, the MIN SAT problem that, given a set of variables and a set of disjunctive clauses, consists of finding a truth assignment that minimizes the number of satisfied clauses. A ρ-approximation algorithm for MIN SAT (with $\rho > 1$) is an algorithm that finds an assignment satisfying at most ρ times the minimal number of simultaneously satisfied clauses.

In [13] an approximability-preserving reduction between MIN VERTEX COVER and MIN SAT is presented transforming any ρ-approximation for the former problem into a ρ-approximation for the latter problem. This reduction can be used to translate any result on the MIN VERTEX COVER problem into a result on the MIN SAT, the number of vertices in the MIN VERTEX COVER instance being the number of clauses in the MIN SAT instance. For instance, the results from [8] for MIN VERTEX COVER lead to the following parameterized approximation result for MIN SAT: *for every instance of* MIN SAT *and for any $r \in \mathbb{Q}$, if there exists a solution for* MIN SAT *satisfying at most k clauses, it is possible to determine with complexity $O^*(1.28^{rk})$ a $2 - r$-approximation of it.*

We also note that the method used in Algorithm 6 can be applied as well to MIN SAT with the following modification of the algorithm. Let $p, q \in \mathbb{Q}$ be such that $p/q = 2\rho - 1$. Build q subsets of variables, each one containing $p/q \times n$ variables (as in Figure 2). For each subset, construct a branching tree, considering only the variables in the subset (the depth of the trees is $p/q \times n$). For each leaf of any of the so-built trees, use some polynomial algorithm with ratio α on the surviving sub-instance. Return the best of the truth assignments computed. ■

The complexity of the modification just described is the same as that of Algorithm 6, i.e., $O^*(2^{n(\alpha-\rho)/(\alpha-1)})$ (the best known ratio is $\alpha = 2$). According to Lemma 1 (that also holds), some of the optimal leaves will have at least $\lambda \geqslant \frac{p}{q}|\text{OPT}|$ clauses satisfied. On the surviving sub-instance, it is possible to satisfy only $(|\text{OPT} - \lambda)$ clauses. Thus, the polynomial algorithm called cannot satisfy more than $\alpha(|\text{OPT}| - \lambda)$ clauses, so that the total number of satisfied clauses will be at most $|\lambda| + \alpha \times (|\text{OPT} - |\lambda|) = \alpha|\text{OPT}| - (\alpha - 1)\lambda \leqslant \alpha|\text{OPT}| - (1 - \alpha)\frac{p}{q}|\text{OPT}|$, deriving an approximation ratio $\alpha - (\alpha - 1)\frac{p}{q} = \rho$.

References

[1] T. Asano and D. P. Williamson. Improved approximation algorithms for MAX SAT. *J. Algorithms*, 42(1):173–202, 2002.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation. Combinatorial optimization problems and their approximability properties.* Springer-Verlag, Berlin, 1999.

[3] G. Ausiello and V. Th. Paschos. Reductions that preserve approximability. In T. F. Gonzalez, editor, *Handbook of approximation algorithms and metaheuristics*, chapter 15, pages 15–1–15–16. Chapman & Hall, Boca Raton, 2007.

[4] R. Battiti and M. Protasi. Algorithms and heuristics for max-sat. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1, pages 77–148. Kluwer Academic Publishers, 1998.

[5] A. Björklund. Determinant sums for undirected Hamiltonicity. In *Proc. FOCS'10*, pages 173–182. IEEE Computer Society, 2010.

[6] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.* To appear in the special issue dedicated to selected papers from FOCS'06.

[7] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation by "low-complexity" exponential algorithms. Cahier du LAMSADE 271, LAMSADE, Université Paris-Dauphine, December 2007. Available at `http://www.lamsade.dauphine.fr/cahiers/PDF/cahierLamsade271.pdf`.

[8] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of combinatorial problems by moderately exponential algorithms. In F. Dehne, M. Gavrilova, J.-R. Sack, and C. D. Tóth, editors, *Proc. Algorithms and Data Structures Symposium, WADS'09*, volume 5664 of *Lecture Notes in Computer Science*, pages 507–518. Springer-Verlag, 2009.

[9] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of MIN COLORING by moderately exponential algorithms. *Inform. Process. Lett.*, 109(16):950–954, 2009.

[10] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of MIN SET COVER by moderately exponential algorithms. *Theoret. Comput. Sci.*, 410(21-23):2184–2195, 2009.

[11] L. Cai and X. Huang. Fixed-parameter approximation: conceptual framework and approximability results. In H. L. Bodlaender and M. A. Langston, editors, *Proc. International Workshop on Parameterized and Exact Computation, IWPEC'06*, volume 4169 of *Lecture Notes in Computer Science*, pages 96–108. Springer-Verlag, 2006.

[12] J. Chen and I. A. Kanj. Improved exact algorithms for MAX SAT. *Discrete Appl. Math.*, 142:17–27, 2004.

[13] P. Crescenzi, R. Silvestri, and L. Trevisan. To weight or not to weight: where is the question? In *Proc. Israeli Symposium on Theory of Computing and Systems, ISTCS'96*, pages 68–77. IEEE, 1996.

[14] M. Cygan, L. Kowalik, and M. Wykurz. Exponential-time approximation of weighted set cover. *Inform. Process. Lett.*, 109(16):957–961, 2009.

[15] M. Cygan and M. Pilipczuk. Exact and approximate bandwidth. *Theoret. Comput. Sci.*, 411(40–42):3701–3713, 2010.

[16] E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev. MAX SAT approximation beyond the limits of polynomial-time approximation. *Ann. Pure and Appl. Logic*, 113:81–94, 2002.

[17] R. G. Downey, M. R. Fellows, and C. McCartin. Parameterized approximation problems. In H. L. Bodlaender and M. A. Langston, editors, *Proc. International Workshop on Parameterized and Exact Computation, IWPEC'06*, volume 4169 of *Lecture Notes in Computer Science*, pages 121–129. Springer-Verlag, 2006.

[18] B. Escoffier and V. Th. Paschos. A survey on the structure of approximation classes. *Computer Science Review*, 4(1):19–40, 2010.

[19] M. Fürer, S. Gaspers, and S. P. Kasiviswanathan. An exponential time 2-approximation algorithm for bandwidth. In *Proc. International Workshop on Parameterized and Exact Computation, IWPEC'09*, volume 5917 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2009.

[20] V. Vazirani. *Approximation algorithms.* Springer, Berlin, 2001.