

A Duality between Clause Width and Clause Density for SAT

Chris Calabro, Russell Impagliazzo, Ramamohan Paturi

University of California, San Diego

9500 Gilman Drive, La Jolla, CA 92093

{ccalabro,russell,paturi}@cs.ucsd.edu *

Abstract

We consider the relationship between the complexities of k -SAT and those of SAT restricted to formulas of constant density.

Let s_k be the infimum of those $c \geq 0$ such that k -SAT on n variables can be decided in time $O(2^{cn})$ and d_Δ be the infimum of those $c \geq 0$ such that SAT on n variables and $\leq \Delta n$ clauses can be decided in time $O(2^{cn})$.

We show that $\lim_{k \rightarrow \infty} s_k = \lim_{\Delta \rightarrow \infty} d_\Delta$. So, for any $\epsilon > 0$, k -SAT can be solved in $2^{(1-\epsilon)n}$ time independent of k if and only if the same is true for SAT with any fixed density of clauses to variables. We derive some interesting consequences from this. For example, assuming that 3-SAT is exponentially hard (that is, $s_3 > 0$), SAT of any fixed density can be solved in time whose exponent is strictly less than that for general SAT.

We also give an improvement to the sparsification lemma of [12] showing that instances of k -SAT of density slightly more than exponential in k are almost the hardest instances of k -SAT. The previous result showed this for densities doubly exponential in k .

1. Introduction

The performance of algorithms or heuristic methods for hard problems, such as NP-complete problems, is often instance-sensitive. This raises the question: how does the difficulty vary with the instance?

In the case of CNF satisfiability, 3 common approaches to parametrically constrain a CNF in an attempt to upper bound how hard it is are

- bound the maximum clause width to some k ,
- bound the ratio of clauses to variables to some Δ ,

- bound the maximum frequency with which variables can appear to some f ,

and we denote the problems with these restricted inputs by, respectively, k -SAT, SAT_Δ , and f -Freq-SAT.

All known upper bounds on the running times of known algorithms for solving these problems in the worst case for particular values of the parameters k, Δ, f are of the form $O(2^{\alpha n})$ where α tends to 1 as the parameter in question tends to ∞ . This raises the question of whether for any of the 3 restriction types there is an algorithm for which α converges to a number less than 1 as the parameter tends to ∞ . (In the case of k -SAT this question is discussed in [18]).

While we do not resolve these questions, we show in this paper that they are equivalent. In fact, we show more: the sequences of best possible exponents (α) for the 3 restriction types have an intuitive functional relationship and even converge to the same limit (theorems 13, 14). In particular, assuming that $\alpha > 0$ for 3-SAT, also called the *exponential time hypothesis* (ETH), the complexities of k -SAT and SAT_Δ are almost the same when $k = \Theta(\lg \Delta)$. We will make this more precise in §4.1.

The intuition for our proofs comes from considering pairs of restrictions. E.g. if we fix k and vary Δ , how does the complexity of $k\text{-SAT}_\Delta = k\text{-SAT} \cap \text{SAT}_\Delta$ vary? If Δ is very small, then the problem will be *underconstrained*, in that there will typically be a very large number of solutions, and we will be able to set many variables arbitrarily without making the problem unsatisfiable. Thus, in some intuitive sense, the problem really involves a much smaller number of variables than it has, and is thus not maximally difficult. On the other hand, an instance where Δ is very big is *overconstrained*, in that setting a few variables usually will force the values of many others. Thus, overconstrained problems are also not the most difficult.

For random formulas, this intuition can be substantiated. The hardest random instances of $k\text{-SAT}_\Delta$ are those where Δ has a threshold value, where the formulas go from satisfiable with high probability to unsatisfiable with high probability. (Interestingly, while it is known that such a threshold function exists, and it is known to asymptotically lie be-

*Research partially supported by NSF Award CCR-0515332, but views expressed are not endorsed by the NSF.

tween two constants, it is not known that the threshold approaches a constant ([9])). This threshold is known to grow exponentially with k ([1, 2]). For densities somewhat below this exponent, there are algorithms that will almost certainly find the satisfying assignments ([10]). On the other hand, for resolution-based methods, random formulas have exponential complexity for constant densities above the threshold, but sub-exponential for every increasing density ([5]). Experimental studies also illustrate how the expected difficulty of the problem peaks at the threshold, then falls gradually off ([14]).

1.1. Relationships to previous work

Our work builds on and extends a line of research on a quantitative theory of the worst-case difficulty of NP-complete problems. This line of work is related to and borrows techniques from parameterized complexity, which looks at the time required to solve instances of NP-complete problems when certain parameters of the problem are small ([7, 8]). However, while parameterized complexity tends to look at the easiest instances of NP-complete problems, those solvable in polynomial-time, exact complexity is intended to reason about the difficulty of the hardest instances.

For example, [12] introduced several techniques for reasoning about the hardest instances of k -SAT and the relative complexities of different NP-complete problems. Their main tool was a *sparsification* lemma, which shows that for any fixed k , there is a density Δ so that k -CNF formulas of density Δ are almost the hardest instances of k -SAT. (Independently, Johnson and Szegegy ([13]) proved a similar result for the maximum independent set problem.) This supports the intuition that the hardest instances have a linear relationship between variables and constraints. However, this constant density was doubly exponential in k . Intuitively, however, the hardest density of k -SAT should be exponential in k , as is the case for random instances. We give a tighter version of the sparsification lemma for which the density is exponential in $O(k \log k)$, which is much closer to the intuitive value.

Using the sparsification lemma, [12] were able to show that the complexities of k -SAT are related, showing, in particular, that $\forall k \geq 3 \ s_k > 0$ is equivalent to ETH (i.e. $s_3 > 0$). So if k -SAT is exponentially hard for any k , then 3-SAT is also exponentially hard.

Assuming ETH, it is natural to ask for a more precise relationship between the complexities of k -SAT for various k . A partial answer was given by [11], who showed that, assuming ETH, k -SAT gets strictly harder as k increases. More precisely, they showed that, for every k there is an l with $s_l \geq s_k + O(1/k)$ (theorem 4). We use this result to prove a similar gap result for constant density SAT (theorem 11).

Besides looking at what the characteristics of the instance reveal about its difficulty, it is also interesting to ask what properties of the set of solutions make a SAT problem easy or hard. An initial result along these lines is found in [6], where they show that solving k -SAT with a promise of unique solutions asymptotically approaches the complexity of general k -SAT, as k increases.

One interesting feature of the above work is that there seems to be a highly fruitful connection between new approaches to SAT algorithms and reductions between restricted versions of SAT. (For example, [11] is largely based on the satisfiability coding lemma of [16], which was originally used to analyze a new SAT algorithm.) Our results are also largely based on a SAT algorithm, a relatively recent algorithm for general SAT due to Schuler [18]. Schuler's approach leverages the performance of k -SAT algorithms ([17, 16, 15]) to solve general SAT through a width reduction technique. We reformulate this technique as a reduction from SAT_Δ to k -SAT.

1.2. Outline

The rest of the paper is organized as follows. In §2 we define the symbols that we use globally throughout the rest. In §3 we state previous results that ours rely upon - in particular the sparsification lemma and Schuler's width-reduction algorithm. In §4 we prove our main results relating k -SAT to both SAT_Δ and f -Freq-SAT. In §5 we show one application: demonstrating the impossibility of sparsifying CNF formulas of size polynomial in the number of variables. In §6 we reduce the sparsification constant of [12] from doubly exponential in k to almost singly exponential. We conclude with a few open questions aimed at further sharpening the picture of the parameterized complexity of SAT.

2. Definitions

CNF is the set of all Boolean formulas in conjunctive normal form. We will consider $F \in \text{CNF}$ to be a set of clauses, and a clause to be a set of non-contradictory literals. We use $|F|$ to represent the number of clauses in F . The variables of F , $\text{var}(F)$, is the set of variables that actually occur in F . The width of F is $\text{width}(F) = \max\{|C| \mid C \in F\}$. The density of F is $\text{density}(F) = \frac{|F|}{|\text{var}(F)|}$. The frequency of F , $\text{freq}(F)$, is the maximum number of times a variable occurs in F . The set of solutions to $F \in \text{CNF}$ is $\text{sol}(F) = \{a \in \{0, 1\}^{\text{var}(F)} \mid F(a) = 1\}$.

$$\begin{aligned} \text{SAT} &= \{F \in \text{CNF} \mid \text{sol}(F) \neq \emptyset\} \\ k\text{-SAT} &= \{F \in \text{SAT} \mid \text{width}(F) \leq k\} \\ \text{SAT}_\Delta &= \{F \in \text{SAT} \mid \text{density}(F) \leq \Delta\} \\ f\text{-Freq-SAT} &= \{F \in \text{SAT} \mid \text{freq}(F) \leq f\}. \end{aligned}$$

For a language $L \in \text{RTIME}(2^{O(n)})$, define

$$c_L = \inf\{c \geq 0 \mid L \in \text{RTIME}(2^{cn})\}.$$

Then define

$$\begin{aligned} s_k &= c_{k\text{-SAT}} \\ d_\Delta &= c_{\text{SAT}_\Delta} \\ \varphi_f &= c_{f\text{-Freq-SAT}}, \end{aligned}$$

and a subscript of ∞ in any of these sequences will represent the limit; e.g. $s_\infty = \lim_{k \rightarrow \infty} s_k$. Occasionally, we will use the same notation when k, Δ are growing functions of the number of variables n ; e.g. $\text{SAT}_\Delta = \{F \in \text{SAT} \mid \text{density}(F) \leq \Delta(|\text{var}(F)|)\}$.

If an algorithm A has several outputs, then we say A has *delay* t if the time between each successive output (and before the first output) is $\leq t$.

3. Previous work

3.1. On SAT constants

The following was shown in [12].

Lemma 1 (Sparsification). *There is a deterministic algorithm A and a function f s.t. $\forall k, \epsilon > 0, F \in k\text{-CNF}$ $A(k, \epsilon, F)$ outputs $F_1, \dots, F_s \in k\text{-CNF}$ s.t.*

1. $s \leq 2^{\epsilon n}$
2. $\text{sol}(F) = \cup_i \text{sol}(F_i)$
3. $\forall i \text{ freq}(F_i) \leq f(k, \epsilon)$
4. each F_i is output with $\text{poly}(n)$ delay, although the degree of the polynomial may depend on k, ϵ
5. $f(k, \epsilon) \in \text{poly}(\frac{1}{\epsilon})$, although the degree of the polynomial may depend on k .

This is far more detail than we need. The following weaker corollary will usually suffice.

Corollary 2. *For each $k, \epsilon > 0$, there is a $O(2^{\epsilon n})$ -time Turing reduction from $k\text{-SAT}$ with n variables to SAT_Δ with n variables, where $\Delta \in \text{poly}(\frac{1}{\epsilon})$, although the degree of the polynomial may depend on k .*

The sequences s_k, d_Δ will be our main object of study. Clearly both are nondecreasing and upperbounded by 1. We will often use the assumption that $s_3 > 0$, also called the *exponential time hypothesis* (ETH) ([12]).

Corollary 3. *The following are equivalent: ETH, $\forall k \geq 3 s_k > 0, s_\infty > 0$.*

Proof sketch. $\text{ETH} \Rightarrow \forall k \geq 3 s_k > 0 \Rightarrow s_\infty > 0$ is obvious. To show $s_\infty > 0 \Rightarrow s_3 > 0$, given a $k\text{-CNF}$ with n variables and m clauses, apply corollary 2 to reduce m to $O(n)$. Then reduce the width of the resulting formula by replacing each k -clause by $O(k^2)$ new 3-clauses in $O(k)$ new variables. The resulting 3-CNF is equivalent to the input $k\text{-CNF}$, but with $O(n)$ variables. \square

The following theorem [11], whose proof uses lemma 1, upperbounds the rate of convergence of s_k .

Theorem 4. $\text{ETH} \Rightarrow \exists d > 0 \forall k \geq 3 s_\infty - s_k \geq \frac{d}{k}$.

3.2. Width reduction

[18] demonstrated a randomized algorithm solving SAT in n variables and m clauses, with no restriction on clause width, in time

$$\text{poly}(n, m) 2^{(1 - \frac{1}{1 + \lg m})n}. \quad (1)$$

This algorithm can be viewed as a Turing reduction from SAT with bounded clause density to SAT with bounded clause width, and we will now present and analyze it as such.

For $k \geq 3$, define the following routine.

- 1 ReduceWidth $_k(F \in \text{CNF})$
- 2 if F has no clause of size $> k$
- 3 output F
- 4 else
- 5 let $C = \{l_1, \dots, l_{k'}\}$ be a clause of F of size $k' > k$
- 6 $C' \leftarrow \{l_1, \dots, l_k\}$
- 7 $F_1 \leftarrow F - \{C\} \cup \{C'\}$
- 8 $F_0 \leftarrow F \uparrow_{\neg l_1, \dots, \neg l_k}$
- 9 /* F_0 is F but with the literals of C' set to false,
- 10 and with true clauses and false literals removed */
- 11 ReduceWidth $_k(F_1)$ /* left branch:
guess C' true */
- 12 ReduceWidth $_k(F_0)$ /* right branch:
guess C' false */

The idea is that the algorithm reduces the width of an arbitrary long clause (size $> k$) C by branching on whether the disjunction C' of the first k literals of C is true or not. If it guesses so, then it replaces C by the short clause C' and

the number of long clauses is reduced by 1. If it guesses not, then it simplifies the formula by eliminating the k variables of C' . Clearly the solutions of the input formula is the union of the solutions of the output formulas.

Let p be some path of length t in the tree T of recursive calls of $\text{ReduceWidth}_k(F)$ and let G be the output at the leaf of p . Let l, r be the number of left, right branches in p .

Then since each right branch eliminates k variables, we have $|\text{var}(G)| = n - kr$ and $r \leq \frac{n}{k}$. Since each left branch eliminates 1 clause of size $> k$, we have $l \leq m$.

So the number of paths in T with r right branches is $\leq \binom{m+r}{r}$, and each outputs a formula with $n - kr$ variables. Furthermore, each output formula is generated with $\text{poly}(n, m)$ delay.

Lemma 5. *If k -SAT can be solved in time $O(2^{sn})$ by some algorithm S then SAT in n variables and $m \geq \frac{n}{k}$ clauses can be solved in time*

$$\text{poly}(n, m)2^{sn + \frac{4m}{2^{sk}}},$$

the polynomial not depending on k, s, S .

Proof. Combining S with ReduceWidth_k gives an algorithm for SAT whose running time is $\leq \text{poly}(n, m)$ times

$$\begin{aligned} & \sum_{r=0}^{\frac{n}{k}} \binom{m+r}{r} 2^{s(n-kr)} \\ & \leq \sum_{r=0}^{m+\frac{n}{k}} \binom{m+\frac{n}{k}}{r} 2^{s(n-kr)} \\ & \leq 2^{sn} (1 + 2^{-sk})^{m+\frac{n}{k}} \\ & \leq 2^{sn} e^{2^{-sk}(m+\frac{n}{k})} \\ & \leq 2^{sn + \frac{4m}{2^{sk}}} \quad \text{since } m \geq \frac{n}{k}. \end{aligned}$$

□

[16] gave an algorithm solving k -SAT in time $\text{poly}(n, m)2^{(1-\frac{1}{k})n}$, where the polynomial does not depend on k , and so Schuler's result (1) can now be obtained from the proof of lemma 5 by taking $s = 1 - \frac{1}{k}, k = 1 + \lg m$. However, we are going to be more interested in using the above as a reduction than to directly design SAT algorithms.

4. Relating the SAT constants

We use the reductions given in the previous section to prove our main results - that the constants for SAT_Δ, k -SAT, f -Freq-SAT are intertwined, and hence the limiting constants are identical.

4.1. Relating k -SAT to SAT_Δ

The proof of $d_\infty = s_\infty$ proceeds in 2 steps. First, sparsification (corollary 2) shows that $s_\infty \leq d_\infty$. To go the other way, we reduce SAT_Δ to k -SAT by using Schuler's width-reduction algorithm. The following lemma bounds the penalty paid for this reduction.

Lemma 6. $\forall \Delta > 0, k \geq \frac{1}{\Delta}, \epsilon > 0 d_\Delta \leq s_k + \epsilon + \frac{4\Delta}{2^{(s_k+\epsilon)k}}$.

Proof. Let S be an algorithm solving k -SAT in time $O(2^{(s_k+\epsilon)n})$. Now take $m = \Delta n$ in lemma 5. □

Taking the limit as $\epsilon \rightarrow 0$ gives the following.

Corollary 7. $\forall \Delta > 0, k \geq \frac{1}{\Delta} d_\Delta \leq s_k + \frac{4\Delta}{2^{s_k k}}$.

By choosing $k = O(\lg \Delta)$ we can make the penalty $\frac{4\Delta}{2^{s_k k}}$ small. In particular, we will want this to be smaller than the distance between s_k and s_∞ , which we can lower bound by using theorem 4. The following technical lemma shows exactly how to choose k .

Lemma 8. $\text{ETH} \Rightarrow \forall \Delta > 0 \exists k \in O(\lg \Delta), \delta \in \Omega(\frac{1}{\lg \Delta}), l$

$$s_l - s_k \geq \frac{4\Delta}{2^{s_k k}} + \delta.$$

Proof. Let d be as in theorem 4. Choose $k = \frac{2}{s_3} \lg(\frac{2}{s_3} \frac{16\Delta}{d}) \in O(\lg \Delta), \delta = \frac{d}{4k} \in \Omega(\frac{1}{\lg \Delta}), l$ so large that $s_\infty - s_l \leq \frac{d}{2k}$. Then

$$\begin{aligned} \frac{d}{16\Delta} 2^{s_k k} & \geq \frac{d}{16\Delta} 2^{s_3 k} = \left(\frac{2}{s_3}\right)^2 \frac{16\Delta}{d} \\ & \geq \frac{2}{s_3} \lg\left(\frac{2}{s_3} \frac{16\Delta}{d}\right) = k. \end{aligned} \quad (2)$$

So

$$\begin{aligned} s_l - s_k & = (s_\infty - s_k) - (s_\infty - s_l) \\ & \geq \frac{d}{k} - \frac{d}{2k} \quad \text{by theorem 4} \\ & = \frac{d}{4k} + \frac{d}{4k} \\ & \geq \frac{4\Delta}{2^{s_k k}} + \delta \quad \text{by (2)}. \end{aligned}$$

□

Now we can upperbound the complexity of SAT_Δ by that of k -SAT for some k (and vice versa) and even add an exponential gap.

Corollary 9. $\text{ETH} \Rightarrow \forall \Delta > 0 \exists l d_\Delta \leq s_l - \Omega(\frac{1}{\lg \Delta})$.

Proof. Combine corollary 7 and lemma 8. □

Corollary 10. $\text{ETH} \Rightarrow \forall k \geq 3 \exists \Delta > 0 s_k \leq d_\Delta - \Omega(\frac{1}{k})$.

Proof. From corollary 2, $s_\infty \leq d_\infty$. Let d be as in theorem 4. Choose Δ so large that $d_\infty - d_\Delta \leq \frac{d}{2k}$. Then

$$\begin{aligned} s_k &\leq s_\infty - \frac{d}{k} \leq (d_\infty - d_\Delta) + (d_\Delta - \frac{d}{k}) \\ &\leq d_\Delta - \frac{d}{2k}. \end{aligned}$$

□

We are now in a position to prove the analog of theorem 4.

Theorem 11. $\text{ETH} \Rightarrow \forall \Delta > 0 \ d_\infty - d_\Delta \geq \Omega(\frac{1}{\lg \Delta})$.

Proof. Combine corollaries 9, 10. □

We need the following to eliminate the assumption of ETH.

Lemma 12. $\neg \text{ETH} \Rightarrow \forall \Delta > 0, k \ d_\Delta = s_k = 0$.

Proof. Use corollary 3 and lemma 6 to conclude that $\forall \Delta > 0, k \geq \frac{1}{\Delta}, \epsilon > 0 \ d_\Delta \leq \epsilon + \frac{4\Delta}{2^{\epsilon k}}$. Choose $k = \frac{1}{\epsilon} \lg \frac{4\Delta}{\epsilon}$ to get that $\forall \epsilon > 0 \ d_\Delta \leq 2\epsilon$, which implies $d_\Delta = 0$. □

Theorem 13. $d_\infty = s_\infty$.

Proof. Combine corollaries 9, 10 and lemma 12. □

It is interesting to note that in [6] it was shown that $s_\infty = \sigma_\infty$, where $\sigma_k = c_{\text{Unique-}k\text{-SAT}}$. So now we have $d_\infty = s_\infty = \sigma_\infty$.

Of course the above theorem did not require the gaps in corollaries 9, 10 that we went through so much trouble to lowerbound. §5 will make use of these gaps.

4.2. Relating k -SAT to f -Freq-SAT

Analogous to theorem 13, we can easily show that $\varphi_\infty = d_\infty = s_\infty = \sigma_\infty$.

Theorem 14. $d_\infty = \varphi_\infty$.

Proof. Since any $F \in \text{CNF}$ with $\text{freq}(F) = f$ has $\text{density}(F) \leq f$, we have $\varphi_f \leq d_f$. So $\varphi_\infty \leq d_\infty$. If $\neg \text{ETH}$, then by lemma 12, $\varphi_\infty \leq d_\infty = 0$. If ETH holds, then corollary 9 and lemma 1 show that $\forall \Delta > 0 \ \exists k, f, \epsilon > 0 \ d_\Delta \leq s_k - \epsilon \leq \varphi_f - \frac{\epsilon}{2}$, and so $d_\infty \leq \varphi_\infty$. □

Combining theorems 14, 11 gives the following.

Theorem 15. $\text{ETH} \Rightarrow \forall f \geq 2 \ \varphi_\infty - \varphi_f \geq \Omega(\frac{1}{\lg f})$.

5. Impossibility of general sparsification

In [18] it was conjectured that c_{SAT} might be < 1 , and it was observed that at least this is true if we restrict SAT instances to have linearly many clauses [4]. (although corollary 9 proves this directly) This naturally makes one wonder whether a more general sparsification is possible, where we assume a polynomial bound on the number of clauses m , but no restriction on the maximum clause width k .

Such a general sparsification lemma would be useful if, say, one had a formula with bounded average clause width, in which case the number of clauses would be polynomially bounded.

Conjecture 16 (general sparsification). \exists function $g \ \forall c, \epsilon > 0 \ \exists$ algorithm $A \ \forall F \in \text{CNF}, |F| \leq n^c, n = |\text{var}(F)| \ A(F)$ outputs $F_1, \dots, F_s \in \text{CNF}$ with $\text{poly}(n)$ delay and where $|\text{var}(F_i)| \leq n$ s.t.

1. $s \leq 2^{\epsilon n}$
2. $\text{sol}(F) \neq \emptyset$ iff $\cup_i \text{sol}(F_i) \neq \emptyset$
3. $\text{density}(F_i) \leq g(c, \epsilon)$
4. $\exists c' > 1 \ g(c', \epsilon) \in 2^{o(\frac{1}{\epsilon})}$. (a weak demand on the rate of growth of g)

We can weaken the demands of the conjecture considerably and still show that it is unlikely to be true. The following is implied by conjecture 16.

Conjecture 17. \exists functions $\alpha(n) \in \omega(1), g(\epsilon) \in 2^{o(\frac{1}{\epsilon})} \ \forall \epsilon > 0 \ \exists$ a $O(2^{\epsilon n})$ -time Turing reduction from SAT_α to $\text{SAT}_{g(\epsilon)}$ that preserves the number of variables.

Theorem 18. $\text{ETH} \Rightarrow$ not conjecture 17.

Proof. Suppose indirectly that conjecture 17 holds. Then

$$\forall \epsilon > 0 \ \exists \Delta \in 2^{o(\frac{1}{\epsilon})} \ d_\alpha \leq d_\Delta + \epsilon,$$

which implies, by corollary 9,

$$\forall \epsilon > 0 \ \exists \delta \in \omega(\epsilon), l \ d_\alpha \leq s_l + \epsilon - \delta.$$

Since $\delta \in \omega(\epsilon)$, we have $\exists \epsilon > 0 \ \epsilon < \delta(\epsilon)$.

So $\exists l \ d_\alpha < s_l$, which implies, by lemma 10,

$$\exists \Delta' > 0 \ d_\alpha < d_{\Delta'} \leq d_\alpha,$$

a contradiction. □

Of course, if you are an optimist, then the contrapositive is the more interesting way to look at the above: if $\text{poly}(n)$ -sized formulas can be sparsified, then k -SAT can be solved fast.

6. Improvement of sparsification

Lemma 1, shown in [12], gave $f(k, \epsilon)$, called the *sparsification constant*, as roughly $(\frac{8k2^k}{\epsilon})^{2^{k-1}}$, which is doubly exponential in k . We will outline how this can be reduced to $(\frac{k}{\epsilon})^{3k}$.

6.1. The algorithm

For fixed $k, \epsilon > 0$, the sparsification algorithm A searches the input formula F for a common subexpression H and a pair of integers c, h such that

- H is a disjunction of $h \geq 1$ literals
- H occurs in $\geq \theta_{c-h}$ clauses of size c

where $\theta_0 \leq \dots \leq \theta_{k-1}$ is a sequence of positive integers depending only on k, ϵ . The c -clauses that contain H form an *admissible* (c, h) -flower of F . A then branches on whether H is true or false.

If A guesses that H is true, then it will insert H into the formula and remove all the clauses that properly contain H , of which there are at least θ_{c-h} of size c .

If A guesses that H is false, then it will remove H from each c -clause in which it occurs and simplify the resulting formula by removing any clause properly containing another.

When the algorithm has several admissible (c, h) -flowers available, it will prioritize lexicographically, selecting an admissible (c, h) -flower with the smallest c and then the largest h . If there is no admissible flower, then A outputs F , since it is already sparsified.

6.2. Old analysis

Say F is c -sparsified iff $\forall c' \leq c$ F has no admissible flower of c' -clauses. If a clause a is added to the formula and is properly contained in another clause b , then b will be removed and we say that a *eliminates* b .

The following salient properties of the algorithm hold no matter how we choose the θ_i , and were shown in [12]:

1. If F is c -sparsified, then each literal occurs in $< \theta_{c-1}$ c -clauses.
2. In any computation path (sequence of branches the algorithm takes), each clause that eliminates any new clauses eliminates $< 2\theta_{c-1} - 1$ c -clauses.

The θ_i are then chosen as $\theta_i = \beta_i \alpha$ where the β_i are defined recursively so that the number of new clauses of size $\leq c$ in any computation path is $\leq \beta_c n$, and α depends only on k, ϵ .

Each branch adds at least 1 new clause, and the number of new clauses in a computation path is $\leq \beta_{k-1} n$. So the number of branches is $\leq \beta_{k-1} n$.

It was also shown in [12] that by choosing $\theta_i = \beta_i \alpha$, the number of branches where the algorithm guesses H to be false (call these *right* branches) is at most $\frac{kn}{\alpha}$. So the number of computation paths is

$$\leq \left(\frac{\beta_{k-1} n}{\frac{kn}{\alpha}} \right). \quad (3)$$

The key observation leading to the choice of β_i is that the number of new clauses of size $\leq c$ in a computation path is

$$\begin{aligned} &\leq \sum_{h=1}^{c-1} \# \text{ new } c\text{-clauses that got eliminated by a new } h\text{-clause} \\ &\quad + \# c\text{-clauses at the end of the computation} \\ &\quad + \# \text{ new clauses of size } \leq c-1. \end{aligned}$$

Using the fact that the summation dominates and property 2, the above is within a constant factor of

$$\sum_{h=1}^{c-1} \theta_{c-1} \beta_h n = \sum_{h=1}^{c-1} \alpha \beta_{c-1} \beta_h n. \quad (4)$$

We need to choose $\beta_c n$ to be at least this, so it suffices for β_c to grow doubly exponentially in c . With this choice of β_c , to keep (3) at most $2^{\epsilon n}$, it suffices to choose α to be roughly exponential in k .

From property 1, the sparsification constant is the sum of the θ_i , which is roughly β_k , and therefore doubly exponential in k .

6.3. New analysis

We claim that property 1 is easily generalized: if F is c -sparsified, then each h -clause occurs in $< \theta_{c-h}$ c -clauses. This allows us to generalize property 2: in any computation path, each h -clause that eliminates any new clauses eliminates $< 2\theta_{c-h} - 1$ c -clauses. The proof of these facts is essentially the same as in [12].

This changes (4) to

$$\sum_{h=1}^{c-1} \theta_{c-h} \beta_h n = \sum_{h=1}^{c-1} \alpha \beta_{c-h} \beta_h n,$$

which is a kind of convolution. We again want $\beta_c n$ to upperbound this. The rate of growth of a sequence whose c th term is the convolution of the first $c-1$ terms with themselves is only exponential, not doubly exponential. (This follows from the convergence of the generating function of the sequence at some positive real.) The extra factor of α in the recursion forces β_c to be $O(\alpha)^c$.

To keep (3) at most $2^{\epsilon n}$, we now choose $\alpha = O(\frac{k}{\epsilon} \lg \frac{k}{\epsilon})$. Our new bound on the sparsification constant is the sum of the θ_i , which is roughly β_k , which is $\leq (\frac{k}{\epsilon})^{3k}$ for ϵ sufficiently small.

6.4. Consequences

From the new sparsification lemma, it follows that instances of k -SAT with relatively low densities are nearly the hardest. Let $\text{SAT}_{k,\Delta}$ be the restriction of k -SAT to formulas of density at most Δ , and $s_{k,\Delta} = c_{\text{SAT},\Delta}$ the corresponding constant. Then by composing the sparsification algorithm with any algorithm for $\text{SAT}_{k,(k/\epsilon)^{3k}}$ we obtain:

Lemma 19. $\forall k, \epsilon > 0 \ s_k \leq s_{k,(k/\epsilon)^{3k}} + \epsilon.$

In particular, letting $\epsilon = k^{-c}$, we obtain:

Corollary 20. $\forall k, c > 0 \ s_k \leq s_{k,2^{3(c+1)k \log k}} + k^{-c}.$

Thus, roughly exponential density formulas are very close to the hardest k -CNF instances.

7. Open problems

Is it possible to further improve the sparsification constant to $(\frac{1}{\epsilon})^{O(k)}$? Is it possible to remove the dependence of the sparsification constant on ϵ ?

References

- [1] D. Achlioptas and C. Moore. The asymptotic order of the k -SAT threshold. *Proc. 43rd IEEE Symposium on the Foundations of Computer Science*, pages 779–788, 2002.
- [2] D. Achlioptas and Y. Peres. The threshold for random k -SAT is $2^k \ln 2 - o(k)$. *Proc. 35th Symposium on the Theory of Computing*, pages 223–231, 2003.
- [3] M. Alekhnovich, E. A. Hirsch, and D. Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *Proc. 31st International Colloquium on Automata, Languages, and Programming*, pages 84–96, 2004.
- [4] V. Arvind and R. Schuler. The quantum query complexity of 0-1 knapsack and associated claw problems. *Proc. 14th International Symposium on Algorithms and Computation*, pages 168–177, 2003.
- [5] P. Beame, R. Karp, T. Pitassi, and M. Saks. On the complexity of unsatisfiability proofs for random k -CNF formulas. *Proc. 30th Symposium on the Theory of Computing*, pages 561–571, 1998.
- [6] C. Calabro, R. Impagliazzo, V. Kabanets, and R. Paturi. The complexity of unique k -SAT: an isolation lemma for k -CNFs. *Proc. 18th IEEE Conference on Computational Complexity*, pages 135–141, 2003.
- [7] R. Downey and M. Fellows. Fixed-parameter intractability. *Structure in Complexity Theory Conference*, pages 36–49, 1992.
- [8] J. Flum and M. Grohe. *Parametrized Complexity Theory*. Springer, 2006.
- [9] E. Friedgut. Necessary and sufficient conditions for sharp thresholds of graph properties, and the k -SAT problem. *J. Amer. Math. Soc.*, 12:1017–1054, 1999.
- [10] A. Frieze and S. Suen. Analysis of two simple heuristics on a random instance of k -SAT. *Journal of Algorithms*, 20(2):312–355, 1996.
- [11] R. Impagliazzo and M. Paturi. Complexity of k -SAT. *Proc. 14th IEEE Conference on Computational Complexity*, pages 237–240, 1999.
- [12] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Proc. 39th IEEE Symposium on Foundations of Computer Science*, pages 653–662, 1998.
- [13] D. Johnson and M. Szegedy. What are the least tractable instances of max independent set? *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 927–928, 1999.
- [14] S. Kirkpatrick and B. Selman. Critical behaviour in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, 1994.
- [15] R. Paturi, P. Pudlak, M. Saks, and F. Zane. An improved exponential-time algorithm for k -SAT. *Proc. 39th IEEE Symposium on Foundations of Computer Science*, pages 628–637, 1998.
- [16] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, 1999. (preliminary version in FOCS’97).
- [17] U. Schoning. A probabilistic algorithm for k -SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.
- [18] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of Algorithms*, 54(1):40–44, 2005.