Sparsification

15 October, 2020 18:12

ETH: $\exists \delta > 0$; 3-SAT cannot be solved in time $2^{\delta n}$

SETH: $\forall \varepsilon > 0 \exists k$; k-SAT — " — $2^{(1-\varepsilon)n}$

- SETH $\Rightarrow$ ETH

Pf: 1st attempt fix $\varepsilon = \frac{1}{2}$ & k s.t. k-SAT cannot be solved in time $2^{\frac{1}{2}n}$.

reduce k-SAT to 3-SAT

ψ ... m clauses, n vars $\longrightarrow$ $(k-2)m$ clauses, $(k-3)m+n$ vars

each clause $(x \vee y \vee z \vee \dots) \longrightarrow (x \vee y \vee a) \wedge (\neg a \vee z \vee \dots)$

k-clause — 3-clause — $(k-2)$-clause

$\vdots$

$\longrightarrow$ 3-clauses, $(k-3)$ new vars $a, b, \dots$

m can be as large as $2^k n^k$

$\Rightarrow$ 3-SAT cannot be solved in time $2^{\frac{n^{1/k}}{10k}}$

(otherwise we would get $2^{\frac{n}{2}}$ time alg for k-SAT)

$\rightarrow$ too weak conclusion

Want: a procedure (sparsification)

k-SAT ψ — m clauses $\longrightarrow$ k-SAT ψ' — $O(m)$ clauses

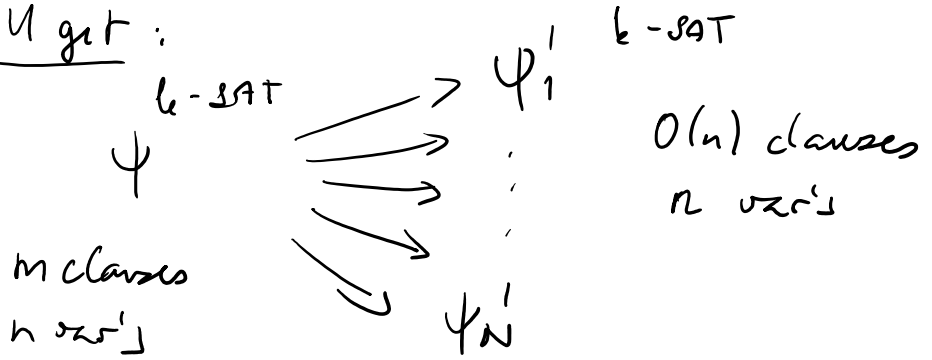
ψ ⟶ ψ'

m clauses, n vars → $O(n)$ clauses, n vars

$\psi \in k\text{-SAT} \iff \psi' \in k\text{-SAT}$

Using such sparsification the above argument would give: 3-SAT cannot be solved in time $2^{\delta n}$ for some $\delta > 0$.

<u>will get</u>:



$\psi \in SAT \iff \exists j \ \psi'_j \in SAT$

$N = 2^{\varepsilon' n}$ for any chosen $\varepsilon' > 0$

so if 3-SAT has $2^{o(n)}$ alg. then k-SAT has $N \cdot 2^{o(n)}$ alg.

set $\varepsilon' = \frac{1}{4}$ → k-SAT alg in time $2^{\left(\frac{1}{4}+o(1)\right)n}$

which contradicts the choice of k & ε in SETH.

so SETH ⇒ ETH .

Sparsification _

Sparsification

idea: $\psi = C_1 \wedge C_2 \wedge C_3 \ldots \wedge C_m$

say $C_1, C_2, \ldots C_s$ contain the same literal ℓ ($= x_i$ or $\neg x_i$) $\quad s \geq \frac{m}{2^k}$

$\psi \rightarrow \psi_1' = \ell \wedge C_{s+1} \wedge C_{s+2} \wedge \ldots C_m$

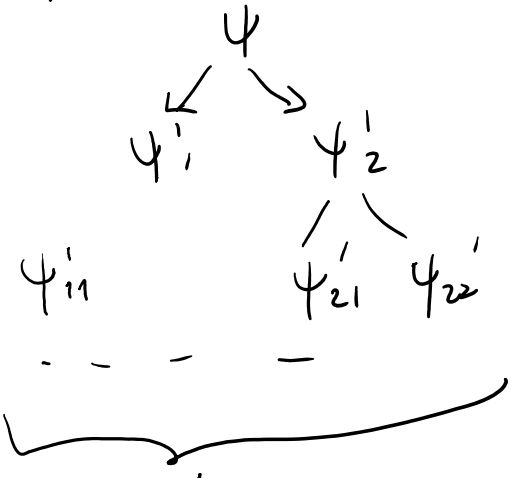
$\psi_2' = (C_1 \setminus \{\ell\}) \wedge (C_2 \setminus \{\ell\}) \ldots (C_s \setminus \{\ell\}) \wedge C_{s+1} \ldots \wedge C_m$

$\psi \in SAT$ iff $\psi_1' \in SAT$ or $\psi_2' \in SAT$

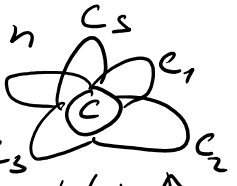
notice: ψ_1' has fewer clauses

ψ_2' has the same # of clauses but instead of k-clauses $\rightarrow (k-1)$-clauses

both ψ_1' & ψ_2' are simpler formulas & we can repeat the process



want $2^{\varepsilon n}$ formulas



sunflower

- instead of picking single literal ℓ pick a clause C s.t. $C \subseteq C_1, C_2, \ldots C_s$

$\psi_1' = C \wedge C_{s+1} \ldots$

$\psi_2' = C_1 \setminus C \wedge C_2 \setminus C \ldots C_{s+1} \ldots$

Actual alg.: fix $\alpha > 1$ s.t. $\frac{\alpha}{\log \alpha} > \frac{4k \cdot 2^k}{\varepsilon'}$

want α formula

Actual alg.: fix $\alpha > 1$ s.t. $\frac{\alpha}{\lg \alpha} > \frac{4k \cdot 2^k}{\varepsilon'}$

define $\beta_i := (4\alpha)^{2^{i-1}-1}$ $\theta_i := \alpha \beta_i$

$i = 1, 2, \ldots k$

notice: θ_i grows with i.

θ_i ... threshold for # of petals (s) of size i

Notice: $\psi = c_1 \wedge c_2 \ldots \wedge c_m$

& $c_i \subseteq c_j$

$\Rightarrow \psi' = c_1 \wedge c_2 \wedge \ldots c_{j-1} \wedge c_{j+1} \wedge \ldots c_m$

$\psi \in SAT$ iff $\psi' \in SAT$ i.e. c_j is redundant

SIMPLIFY(ψ): remove all clauses that are superset of some other clauses.

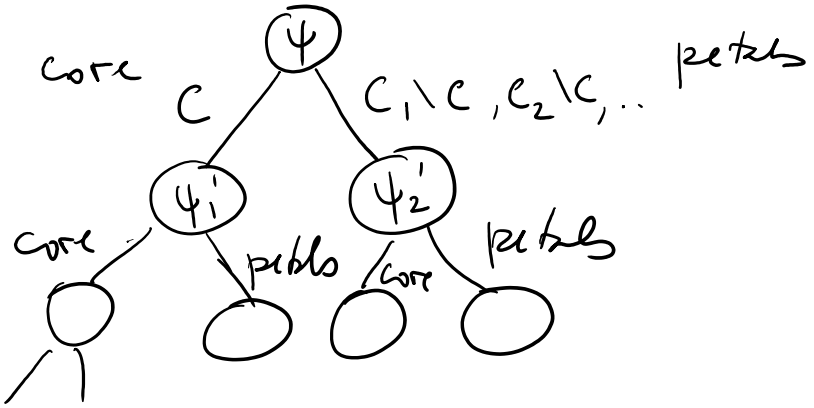
REDUCE(ψ):

1) $\psi \leftarrow$ SIMPLIFY(ψ)
2) for $j = 2$ to k do
   for $i = 1$ to $j-1$ do
   if there are j-clauses $c_1, c_2, \ldots c_t$ and $(j-i)$-clause C s.t. $C \subseteq c_1 \cap c_2 \ldots \cap c_t$ where $t \geq \theta_i$
   then
   REDUCE($\psi \setminus c_1 \ldots c_t \cup C$)
   REDUCE($\psi \setminus c_1 \ldots c_t \cup C \setminus c_1 \; C \setminus c_2 \ldots C \setminus c_t$)

Searching for a sunflower in a specific order.

REDUCE($\psi \setminus c_1 \dots c_t \cup c$)
REDUCE($\psi \setminus c_1 \dots c_t \cup c_1 \setminus c, c_2 \setminus c, \dots c_t \setminus c$)
return;

no large enough sunflower → 3) output (ψ)



Claim: For any output for ψ', $\forall$ literal ℓ, $\forall i \in \{2, \dots, k\}$
ℓ appears at most $(\theta_{i-1} - 1)$ in i-clauses of ψ'.

Pf: If not, we would get a sunflower with
core $C = \{\ell\}$ and $\geq \theta_{i-1}$ petals
$\rightarrow$ we could reduce ψ' further ⊠

$\Rightarrow$ any output ψ' contains at most $k \cdot \theta_k$ times each literal

$\Rightarrow$ #clauses $\leq \underbrace{2n}_{\#\text{literals}} \cdot k \cdot \theta_k = O(n)$.

Claim: $\forall i < k$, once we add an i-clause during

the algorithm, for all descendant Formulas each literal from any i-clause appears at most $2\theta_{i-1}$-times in i-clauses.

Pf: When an i-clause is added, no literal can appear more than $(\theta_{i-1}-1)$-times in i-clauses already present as o/w the literal would form a core of a sunflower which would have to be chosen before sunflower consisting of j-clauses where $j > i$.

- If the added i-clause is a core, then occurrence of literals in i-clauses increased by at most 1 and $(\theta_{i-1}-1)+1 \leq 2\theta_{i-1}$.
- If the added i-clause is a petal, then among the petals, no literal can appear more than $(\theta_{i-1}-1)$-times as o/w we would have to add it into the core of the sunflower. In the case, no literal can appear more than $\theta_{i-1}-1+\theta_{i-1}-1 \leq 2\theta_{i-1}$ times.
- In all descending Formulas the property is preserved as we can only remove i-clauses during SIMPLIFY() or add new i-clauses, but then the above argument applies. ∎

Claim: A given clause C can eliminate at most $2\Theta_{i-1}$ added i-clauses, $i<k$, during call to SIMPLIFY().

Pf: When C eliminates some clauses during SIMPLIFY(), it eliminates all its supersets. New clauses are always subsets of existing clauses so after the elimination, no new supersets of C can be added to the formula. So C eliminates its supersets only once (during a particular branch of the recursion.)

Let $l \in C$ be a literal. At the point of elimination of supersets of C, l occurs in at most $2\Theta_{i-1}$ added i-clauses (by previous claim). Hence, C eliminates at most $2\Theta_i$ i-clauses. ∎

Claim: Along a particular branch of the recursion, $\forall j<k$ we can add at most $2\beta_j n$ i-clauses, $i \le j$.

Pf: <u>induction on j</u>:

$j=1$: There are $2n$ distinct literals forming a unit clause, each of them can be added at most once (as SIMPLIFY() eliminates all its supersets)

$j>1$ since $\beta_1 = 1$ the claim follows for $j=1$.

$j-1 \to j$: There are $2n$ literals, each of them

occurs at most θ_{j-1}-times in j-clauses present in the final formula.
Other added j-clauses must have been eliminated by some i-clause, $i<j$, during SIMPLIFY(). Each i-clause can eliminate $\leq 2\theta_{j-1}$ added j-clauses.

Thus, the # of i-clauses, $i \leq j$, added is bounded by:

(I.H.) (remaining j clauses) (eliminated j-clauses)

$$2\beta_{j-1}n + \theta_{j-1}\cdot 2n + \underbrace{2\beta_{j-1}n}_{\text{(I.H.)}} \cdot 2\theta_{j-1}$$

$$\leq 2n\left(\beta_{j-1} + \theta_{j-1} + 2\beta_{j-1}\theta_{j-1}\right) := (*)$$

As $\beta_j = (4\alpha)^{2^{j-1}-1}$ & $\theta_{j-1} = \beta_{j-1}\alpha$

$$\Rightarrow \beta_j = 4\alpha\,\beta_{j-1}^2,$$

thus $(*) \leq 2n\,\beta_j$. ☐

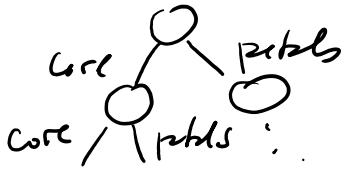
Claim: Along any branch of the recursion we add petals at most $\frac{2\beta_k n}{\alpha}$ - times

Pf: $\forall i < k$, we add at most $\beta_i 2n$ i-clauses when we add petals, we add at least θ_i of them at once $\Rightarrow$ this can be done at most $\frac{2n\beta_i}{\theta_i} = \frac{2n}{\alpha}$ times.

Summing up over $i \Rightarrow \frac{2nk}{\alpha}$ ☐

- To bound the # of branches ($= \alpha^{\text{output formulas}}$):

During the recursion we take at most $\frac{2nk}{\alpha}$ turns right to add petals.



. . . . . .

This is from among $\leq 2\beta_{k-1} n$ additive op's.

$$\#\text{branches} \leq \binom{2\beta_{k-1} n}{\frac{2nk}{\alpha}} \leq \left(\frac{2\beta_{k-1} n \cdot e}{2nk/\alpha}\right)^{2nk/\alpha}$$

$$\leq \left(\frac{\beta_{k-1} \cdot e \cdot \alpha}{k}\right)^{2nk/\alpha}$$

$$\leq \left(\left(4\alpha\right)^{2^{k-1}}\right)^{2nk/\alpha}$$

$$\leq (4\alpha)^{2^k nk/\alpha}$$

$$\leq 2^{n \frac{2^k \cdot k \cdot \log 4\alpha}{\alpha}} \overset{\alpha > 1}{\leq} 2^{n \frac{2^k \cdot k \cdot 4 \log \alpha}{\alpha}}$$

$$\leq 2^{\varepsilon' n}$$

□