1 Sparsification

Recall the Exponential Time Hypothesis (ETH) of [1]: $\exists \delta > 0$ such that 3-SAT requires $2^{\delta n}$ time.

ETH is a strengthening of P $\neq$ NP: not only does 3-SAT require super-polynomial time, but it requires exponential time.

Now, when you saw NP-hardness reductions in prior algorithms classes, there were many of them from 3-SAT. Could we use these, and by replacing P $\neq$ NP as an assumption by ETH, obtain that a variety of other problems require exponential time?

Let us explore this question by revisiting the standard reduction from 3-SAT to Independent Set.

The standard reduction from 3-SAT to Independent Set. Recall the Independent Set (IS) Problem: Given a graph $G = (V, E)$ and an integer k, is there a subset $I \subseteq V$ with $|I| \geq k$ so that for every $(u, v) \in E$, either $u \notin I$ or $v \notin I$ (or both), i.e. I is independent.

Let us recall how to reduce 3-SAT to IS. Given a 3-CNF formula F with m clauses and n variables, we produce a graph as follows. For every clause C_j with literals ℓ_1, ℓ_2, ℓ_3, we create three nodes, one for each literal. Let's call the node corresponding to literal ℓ_k appearing in clause C_j, $v(C_j, \ell_k)$. We make the three nodes corresponding to each clause into a triangle: connect every two vertices corresponding to the same clause by an edge.

Then, consider every variable x_k of F. If x_k appears as x_k in clause C_j and as $\neg x_k$ in some other clause C_t, then connect $v(C_j, x_k)$ and $v(C_t, \neg x_k)$ by an edge.

In other words, there are two types of edges: those connecting nodes for the same clause, and those connecting nodes for the same variable occurring negated and not-negated.

We will show that this graph G contains an independent set of size m if and only if F is satisfiable.

Now, suppose that F is satisfied by some assignment σ. Assignment σ selects (at least) one literal per clause to set to true. Let $\ell(j)$ denote that literal for clause C_j. Consider $I = \{v(C_1, \ell(1)), \ldots, v(C_m, \ell(m))\}$. I has m vertices of G, one for each clause. Let's see why I is independent. There are no edges of the first type since there are no two nodes for the same clause. Suppose then that there is an edge in G between $v(C_i, \ell(i))$ and $v(C_j, \ell(j))$. But then, $\ell(i) = \neg\ell(j)$ since it must be an edge of the second type. This is a contradiction since the assignment σ cannot set both $\ell(j)$ and $\neg\ell(j)$ to true.

Now suppose that G has an independent set I of size m. I can have at most one vertex $v(C_j, \odot)$ for each clause C_j since any two vertices for the same clause are connected by an edge. Since I has size m, it must actually have exactly one vertex per clause. Let $v(C_j, \ell(j))$ be the node for clause C_j. Then we create an assignment for the variables of F as follows: for every $\ell(j)$, let the assignment be such that $\ell(j)$ is set to true. Notice that since there are no $\ell(j)$ and $\ell(i)$ for which $\ell(j) = \neg\ell(i)$, the assignment to the variables corresponding to the literals $\ell(j)$ are consistent. To complete this to a full assignment, assign 1 to all variables that do not appear in the literals $\ell(j)$. We get an assignment that sets at least one literal to true in each clause, and hence F is satisfiable.

An example IS instance is shown in Figure 1. The triangles in the figure correspond to the clauses.

Independent Set requires exponential time, under ETH. Let's attempt to use the above reduction to show that IS requires exponential time under ETH. Our supposed proof would proceed by contradiction. Suppose that for every $\delta > 0$, there is an $O(2^{\delta N})$ time algorithm for IS in N-node graphs. Now, given a 3-CNF F on n variables and m

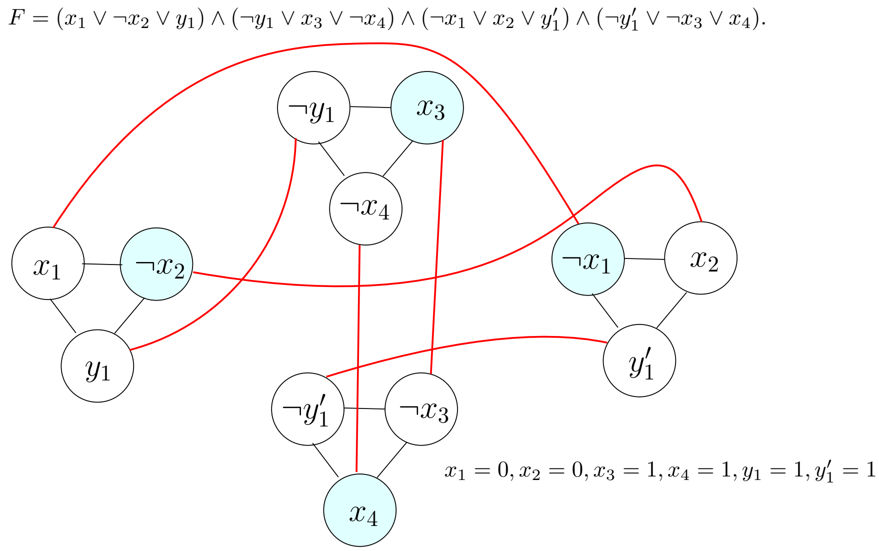

Figure 1: The reduction from 3-SAT to IS for a particular formula F. The corresponding Independent Set is shown, together with the corresponding satisfying assignment of F.

clauses, we want to use this algorithm to solve F in $O(2^{\varepsilon n}\text{poly}(m))$ time for all $\varepsilon > 0$. We pick our favorite $\varepsilon > 0$ and we want to pick the δ as a function of ε for the $2^{\delta N}$ time IS algorithm.

We use the reduction to form a graph G from F. The number of nodes N of G is $3m$ as we have three nodes per clause. So for the δ we pick, we would be solving 3-SAT in time $O(2^{\delta \cdot 3m})$. Now since m can be as large as n^3, there can be no constant δ that we can pick to get an $2^{\varepsilon n}\text{poly}(m)$ time algorithm for 3-SAT!

What we would like is a reduction that produces an IS instance graph on $O(n)$ vertices instead of $O(m)$. Then, our approach would work. Suppose that we could get a graph G out of a 3-SAT formula F so that G has $N \leq cn$ nodes when F has n variables. Then, for every $\varepsilon > 0$, we set $\delta = \varepsilon/c > 0$ and use the supposed $O(2^{\delta N})$ time IS algorithm to solve F in time $O(2^{\varepsilon n})$ due to our choice of δ.

One way to achieve such a reduction would be to *sparsify* 3-SAT, i.e. reduce 3-SAT on n variables and an arbitrary number of clauses to 3-SAT on $O(n)$ variables and $O(n)$ clauses. Then, we can just use the reduction we already have. Such a reduction is not known. In fact, if such a reduction could be carried out in polynomial time, then coNP would be in NP/poly, something that we do not believe is true [5] (actually via [4] even sparsifying to a single formula on $O(n^{3-\epsilon})$ number of clauses for any $\epsilon > 0$ would imply the same conclusion; more on the limits of sparsification can be found in [6]).

Fortunately, an almost as good sparsification is known. It is an algorithm that takes an arbitrary k-CNF (for any $k \geq 3$) formula on n variables (possibly with a huge number of clauses) and produces a *family* of formulas with a linear number of clauses. Specifically, the following "Sparsification Lemma" was proven by Impagliazzo, Paturi and Zane (2001) [3]:

Theorem 1.1 ("Sparsification Lemma") *Let $\varepsilon > 0$, $k \geq 3$ constant. There is a $2^{\varepsilon n} \cdot poly(n)$ time algorithm that takes a k-CNF F on n variables and produces $F_1, \ldots, F_{2^{\varepsilon n}}$, $2^{\varepsilon n}$ k-CNFs such that F is satisfied if and only if $\bigvee_i F_i$ is satisfied and each F_i has n variables and $n \cdot (\frac{k}{\varepsilon})^{O(k)}$ clauses. In fact, each variable is in at most poly$(\frac{1}{\varepsilon})$ clauses, and the F_i are over the same variables as F.*

We won't prove the theorem above, but at the end of the notes we will give some intuition for the proof.

Let's denote $c(k, \varepsilon) := (\frac{k}{\varepsilon})^{Ck}$ for C large enough for the Sparsification Lemma statement.

We prove a simple but important corollary:

Corollary 1.1 *Under ETH, there exist constants $\varepsilon, \delta > 0$ s.t. 3-SAT on n variables and $m = c(3,\varepsilon)n$ clauses requires $2^{\delta m}$ time.*

Proof. Suppose for contradiction that for every $\varepsilon > 0$ and every δ, 3-SAT on $m \leq c(3,\varepsilon)n$ clauses is in $O^*(2^{\delta m})$ time. Here recall that $c(3,\varepsilon) = (3/\varepsilon)^{3C}$.

Pick any $\delta' > 0$. Let F be a 3-SAT formula on n variables. We will show how to solve its satisfiability problem in $O^*(2^{\delta' n})$ time.

Set $\varepsilon = \delta'/2$ and let $\delta = \delta'^{3C+1}/(2 \cdot 6^{3C}$.

Apply the algorithm from the Sparsification lemma on F to obtain $2^{\varepsilon n}$ formulas F_i on n variables and $m \leq c(3,\varepsilon)n$ clauses. Then use the fast algorithm from our assumption to solve SAT for every F_i in time $O^*(2^{\delta m})$ time.

This solves SAT for F in total time, within polynomial factors,

$$2^{\varepsilon n} \cdot 2^{\delta c(3,\varepsilon)n} \leq 2^{n(\varepsilon + \delta(3/\varepsilon)^{3C})}.$$

The exponent above is $n(\varepsilon+\delta(3/\varepsilon)^{3C}) = n(\delta'/2+\delta(6/\delta')^{3C})$. By setting $\delta = \delta'^{3C+1}/(2\cdot 6^{3C}$ the exponent becomes $\delta' n$ and we have solved SAT on F in $O^*(2^{\delta' n})$ time for all $\delta' > 0$, contradicting ETH. □

Now, let's see how to use Theorem 1.1, and in particular its corollary, together with the 3-SAT to IS reduction to show that IS requires exponential time under ETH.

Assume that IS on N node graphs can be solved in $O^*(2^{\delta N})$ time for all $\delta > 0$.

Let's show how to solve 3-SAT on a formula F with n variables and $m = c(3,\varepsilon)n$ clauses in $2^{\delta' m}$ time for all $\delta' > 0$.

Reduce 3-SAT on F via the standard reduction to IS to obtain a graph G on $N = 3m = 3c(3,\varepsilon)n$ vertices. Solve IS on G using the supposed algorithm in $O^*(2^{\delta N})$ time for $\delta = \delta'/3$. This solves 3-SAT on F in time $O^*(2^{\delta' m})$, contradiction ETH via the corollary of the Sparsification Lemma.

2 SETH implies ETH.

Recall **SETH**: For every $\varepsilon > 0$ there is a k so that k-SAT on n variables cannot be solved in $O(2^{(1-\varepsilon)n})$ time.

Does SETH imply ETH?

To address this question, let us consider a more believable hypothesis than ETH:

More Believable ETH (MBETH): There exists a $k \geq 3$ and a $\delta > 0$ so that k-SAT on n variables cannot be solved in $O(2^{\delta n})$ time.

Clearly, ETH implies MBETH. Similarly, SETH implies MBETH: SETH implies for instance that there is a k that cannot be solved in $O(2^{0.8n})$ time (choosing $\varepsilon = 0.2$) which clearly implies MBETH.

We will actually show that MBETH is equivalent to ETH! If there is any k for which k-SAT requires exponential time, then 3-SAT also does. As a byproduct we get that SETH implies ETH.

To prove this, we use Sparsification again.

We will show that if 3-SAT is in $2^{\delta n}$ time for all $\delta > 0$, then k-SAT is also in $2^{\delta n}$ time for all δ.

We begin with F, a k-CNF with n variables and m clauses.

Consider the standard reduction from k-SAT to 3-SAT. Given a k-CNF F we perform the following conversion: for each clause $(x_1 \vee \ldots \vee x_k)$, where the x_i are literals, replace it with $(x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\neg y_{k-3} \vee x_{k-1} \vee x_k)$. Each original clause gives rise to $k-3$ new variables and $k-2$ clauses, giving a 3-CNF with $n + m(k-3) \leq mk$ variables and $\leq mk$ clauses. Now, armed with this reduction, we take a k-CNF F and first sparsify F via Theorem 1.1. Then we apply our k-CNF to 3-CNF transformation. This yields $2^{\varepsilon n}$ k-CNFs with $O(n)$

clauses, and then $2^{\varepsilon n}$ 3-CNFs but with only $O(nk)$ variables and clauses. We solve these CNFs with our supposedly fast algorithm for 3-SAT.

Our final running time is $2^{\varepsilon n} \cdot 2^{O(\delta nk)}$, so we just choose $\varepsilon = \varepsilon'/2$ and $\delta \approx \frac{\varepsilon'}{2kc}$, where c is the constant in the big-Oh $O(\delta nk)$. This leads to a $2^{\varepsilon' n}$ algorithm for k-SAT, as desired.

3 K-SUM

Recall that the K-SUM problem is as follows: Given a set S of n integers and a target integer T, determine whether S contains K integers $a_1, \ldots, a_K$ so that $\sum_{i=1}^{K} a_i = T$. We can solve K-SUM in time $O(n^{\lceil K/2 \rceil})$ via a "meet-in-the-middle" approach and this is the best known running time. We will see this in later lectures; the algorithm is very similar to the Subset Sum algorithm in the last lecture! We will now show that under ETH the exponent of $\Omega(k)$ is necessary.

Theorem 3.1 *If* $\forall \varepsilon > 0 \exists k$ *such that* k-SUM *on "small" numbers is in* $O(n^{\varepsilon k})$ *time, then ETH is false.*

The result is due to Patrašcu and Williams [7].

Proof. We will use the Corollary of the Sparsification Lemma again.

Let F be a 3-CNF with n variables and $m = c(3, \varepsilon)n$ clauses. We assume via the Corollary that solving 3-SAT on F requires $2^{\delta m}$ time for some δ.

We first convert F to F' which is a "1 in 3 SAT" instance (exactly one literal must be true in each clause). We do this by replacing each clause $(x \vee y \vee z)$ with $(x \vee a \vee d) \wedge (y \vee b \vee d) \wedge (a \vee b \vee e) \wedge (c \vee d \vee f) \wedge (z \vee c)$, where a, b, c, d, e, f are variables that only appear in these 5 clauses corresponding to the original clause $(x \vee y \vee z)$.

It is not hard to see that the clause $(x \vee y \vee z)$ is satisfied by an assignment ϕ to x, y, z iff the 5 clauses corresponding to the clause can be satisfied in a 1-in-3 manner by tacking onto ϕ an assignment to $\{a, b, c, d, e, f\}$.

This yields 6 new variables per clause and 5 clauses per clause, so F' has $n' = O(m)$ variables and $m' = O(m)$ clauses. Let $n' = Cm$ for some constant C, for concreteness.

Now we will create an instance of k-SUM. Partition the variables of the 1-in-3 formula into k groups of n'/k size each: $V_1, \ldots, V_k$. Look at all possible $2^{n'/k}$ partial assignments for each group. We will assign a number to each partial assignment for each group, written in base $(k+1)$. The number corresponding to the partial assignment ϕ has a section corresponding to its group and a section corresponding to clauses it satisfies. In the group section, there are k positions, and there is a 1 in position i for the V_i that ϕ corresponds to and a 0 otherwise. The target t has a 1 for each digit in this section, forcing the numbers chosen to solve the k-SUM problem to use one assignment from each group.

For the clause section, record the number of literals for each clause j that ϕ sets to true (if there is more than one for any clause, omit ϕ since it satisfies too many already). The target t has 1 for the digit corresponding to each clause j.

The numbers have $m'+k$ components in base $k+1$, so their size is $(k+1)^{m'+k}$ and when $m' = O(m)$ this is $\approx k^{O(m)}$ size. The number of numbers is $N = k2^{n'/k}$ so that the size of the bit representation of the numbers, $O(\log(k^{n'}))$, is $O(k \log k \log N)$.

If k-SUM on N, $O((k \log k) \log N)$-bit numbers is in $O(N^{\delta k})$ time $\forall \delta$, then our resulting instance can be solved in $O((k2^{n'/k})^{\delta k})$ time, which is $k^{O(k)} 2^{\delta n'} = k^{O(k)} 2^{\delta C m}$ time.

Hence by setting for every $\delta' > 0$, $\delta = \delta'/C$, our 1-in-3 SAT instance, and also our original 3SAT instance, can be solved in $k^{O(k)} 2^{\delta m}$ time for all δ, refuting ETH.

□

4 Sparsification Revisited

We now outline how the sparsification algorithm in Theorem 1.1 works.

Suppose we have the two clauses $c_1 = (x \vee y \vee z)$ and $c_2 = (x \vee y' \vee z')$. If x is true, we can remove both clauses c_1 and c_2. If x is false, we can replace the two with $(y \vee z)$ and $(y' \vee z')$.

This method generalizes to finding a "weak-sunflower", a set of clauses that share some common sub-clause (weak because the petals can intersect). Either they can all be removed and replaced with the subclause or this common subclause can be removed from each clause. There is a tradeoff between the number of clauses involved and the size of the subclause.

If a k-CNF formula F on n variables has $> cn$ clauses, then consider the literal x that appears in the maximum number of clauses q. Then the number of clauses must be at most $2nq$. So x must appear in $> c/2$ clauses. Thus any such dense enough formula must contain a weak-sunflower on $> c/2$ clauses with a "heart" containing a literal common to all sunflower clauses. Thus, if no large weak sunflower can be found, the formula has become sparse.

The sparsification algorithm goes through the clauses by size: iterate over i from 2 to k and then j from 1 to $k-1$ (i is the size of clauses being looked at, and j corresponds to the size of the petals). Find ℓ_i i-clauses that intersect in $(i-j)$ literals. In one case remove the clauses and add the core as a clause; in the other case, set all literals in the core to false (and shrink the ℓ_i clauses into j-clauses).

Branch like this for depth εn, yielding $2^{\varepsilon n}$ leaves. Each one of these is one of the output formulas. A careful analysis states that they must be sparse.

References

[1] Russell Impagliazzo, Ramamohan Paturi: On the Complexity of k-SAT. J. Comput. Syst. Sci. 62(2): 367-375 (2001)

[2] Chris Calabro, Russell Impagliazzo, Ramamohan Paturi: On the Exact Complexity of Evaluating Quantified k -CNF. Algorithmica 65(4): 817-827 (2013)

[3] Russell Impagliazzo, Ramamohan Paturi, Francis Zane: Which Problems Have Strongly Exponential Complexity? J. Comput. Syst. Sci. 63(4): 512-530 (2001)

[4] Holger Dell, Dieter van Melkebeek: Satisfiability Allows No Nontrivial Sparsification unless the Polynomial-Time Hierarchy Collapses. J. ACM 61(4): 23:1-23:27 (2014)

[5] Lance Fortnow, Rahul Santhanam: Infeasibility of instance compression and succinct PCPs for NP. J. Comput. Syst. Sci. 77(1): 91-106 (2011)

[6] R. Santhanam and S. Srinivasan: On the Limits of Sparsification. In: Czumaj A., Mehlhorn K., Pitts A., Wattenhofer R. (eds) Automata, Languages, and Programming. ICALP 2012. Lecture Notes in Computer Science, vol 7391, 2012. Springer, Berlin, Heidelberg.

[7] Mihai Patrašcu, Ryan Williams: On the Possibility of Faster SAT Algorithms. SODA 2010: 1065-1075.