COMS E6998-15: Fine-grained complexity (Spring'26) January 27, 2026

Lecture 2: Reductions & SETH

Instructor: *Josh Alman* Scribes: *Innokentiy Kaurov, Pires William (2022)*

1 Fine Grained Reductions

Last lecture we used two reductions, one to prove that SETH implies OVC, and another to show that OVC implies that we can't compute the diameter of a sparse undirected graph in time $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$. Looking at the second reduction, we realize that it is important that the construction of the graph doesn't take quadratic time, otherwise this wouldn't give an algorithm solving OV in time $\mathcal{O}(n^{2-\varepsilon})$. Therefore, we can't work with Cook nor Karp reductions, as they only require the reduction to take polynomial time in the size of the input. We need a more refined definition, that is why we introduce fine grained reductions.

Definition 1.1. *Let A,B be computational problems, let $a(n)$, $b(n)$ be two functions $\mathbb{N} \to \mathbb{N}$. We say that problem A (a, b)-reduces to B, written $A \leq_{a,b} B$, if for every constant $\varepsilon > 0$, there is a constant $\delta > 0$ such that there is an algorithm R for A with the following properties:*

- *R runs in time $\mathcal{O}(a(n)^{1-\delta})$ on inputs to A of length n.*
- *R makes q calls to an oracle for B, where each query has size $n_1, n_2, \ldots, n_q$, such that:*

$$\sum_{i=1}^{q}(b(n_i))^{1-\varepsilon} \leq \mathcal{O}(a(n)^{1-\delta})$$

This definition provides a framework for showing that if we had an algorithm for B running in time $\mathcal{O}(b(n)^{1-\varepsilon})$, then we could solve A in time $\mathcal{O}(a(n)^{1-\delta})$. In particular, the second property of R guarantees us that, if we were to replace the oracle calls in R by the algorithm for B, the running time would still be $\mathcal{O}(a(n)^{1-\delta})$. Such a reduction shows that a polynomial speedup for B would give a polynomial speedup for A.

Theorem 1.1. *If A cannot be solved in time $\mathcal{O}(a(n)^{1-\delta})$ for any $\delta > 0$ and $A \leq_{a,b} B$, then B cannot be solved in time $\mathcal{O}(b(n)^{1-\varepsilon})$ for any $\varepsilon > 0$.*

Extensive work has been done, using fine grained reductions combined with OVC, to show similar results for other problems, examples include problems motivated by applications in Computational Biology (Longest Common Subsequence, Local Alignment, ...). Other types of problems whose hardness can be based on OVC include Dynamic Programming problems, Nearest Neighbor Search, String problems, and Machine Learning problems. Finally, OVC has also been used to show hardness of approximation for different problems. For instance, the reduction from the Orthogonal Vectors problem to Graph Diameter exhibits that, assuming OVC, we can't distinguish graphs with diameter 2 from graphs with diameter 3 in time $\mathcal{O}(n^{2-\varepsilon})$ for any constant $\varepsilon > 0$.

Besides, during the course, we also employ two additional hardness assumptions, one for the 3SUM problem and another for the APSP problem.

3SUM.
Given. A set of integers S of size n.
Task. Decide whether there exist $x, y, z \in S$ such that $x + y + z = 0$.

The problem can easily be solved in time $\mathcal{O}(n^3)$ by looking at all triples of numbers in S and verifying whether or not they sum to 0. A more careful look, left as an exercise to the reader, shows that there is an algorithm running in time $\mathcal{O}(n^2)$ for the problem.

The 3SUM conjecture states that there is no algorithm for 3SUM running in time $\mathcal{O}(n^{2-\varepsilon})$ for any constant $\varepsilon > 0$. Assuming this conjecture, hardness results in the form of reductions from 3SUM have been explored extensively in the literature. Key examples include several problems in Computational Geometry, which find applications in robotics, as well as problems related to triangles in graphs.

The third main conjecture we are going to employ in the course relates to the APSP problem.

All-Pairs Shortest Path.
Given. A weighted directed graph with n vertices.
Task. Output the distances between all pairs of vertices.

The most basic algorithm (e.g. Floyd-Warshall) runs in time $\mathcal{O}(n^3)$. There has been substantial effort to improve this runtime by logarithmic factors, and a recent work by Williams [Wil13] achieved a complexity of $\frac{n^3}{2^{\Omega(\sqrt{\log n})}}$, which is the best result to date. Still, no algorithm with a polynomial improvement has been found, which motivates the APSP conjecture.

Conjecture 1.1 (The APSP Conjecture)**.** *There is no algorithm for* APSP *running in time $\mathcal{O}(n^{3-\varepsilon})$ for any constant $\varepsilon > 0$.*

Hardness results assuming the APSP conjecture have been demonstrated for several problems. Examples include finding the radius of a dense graph and detecting the existence of a negative triangle in a weighted graph. Interestingly, for many problems it has been shown that not only does APSP reduce to them, but they reduce to APSP too, such as the radius problem. This defines a family of problems such that either all of them have an algorithm running in time $\mathcal{O}(n^{3-\varepsilon})$ for some constant $\varepsilon > 0$, or none do.

A problem worth noting is the Triangle Collection problem, whose hardness can be deduced assuming any of SETH, 3SUM conjecture or APSP conjecture, as shown in [AWY15].

Triangle Collection.
Given. An undirected graph with n vertices and a coloring of its vertices with $\leq n$ colors.
Task. Determine whether there exists a triplet of colors a, b, c such that no triangle is colored with these colors.

A holy grail of fine-grained complexity would be to show that one can actually rely on a singular hardness conjecture in order to show the hardness of all the above interesting problems, and many more. For example, showing a fine-grained reduction between APSP and 3SUM would be a very significant step towards that goal. Alas, at this point, not only are such reductions unknown, but there is some evidence to support the claim that coming up with such reductions might be a hard task.

2 SETH

As many of the reductions we shall see in the course assume SETH, there is merit in discussing the plausibility of SETH and it is important to note that there is no consensus in the research community on whether the hypothesis should be true or not.

Note that SETH is formulated in terms of k, instead of just ε, because k-SAT problems are in fact solvable in time exponentially better than $\mathcal{O}(2^n)$. For example, 3-SAT has an exact algorithm running in time $\mathcal{O}(1.32793^n)$ [Liu18].

In exploring whether or not SETH is true, we might first ask ourselves why it is called the **Strong** Exponential Time Hypothesis. It turns out SETH is a strengthening of the following hardness conjecture:

Conjecture 2.1 (Exponential Time Hypothesis, ETH)**.** *There exists a constant $\delta > 0$ such that 3-SAT requires time $\Omega(2^{\delta n})$.*

In particular, ETH rules out the possibilities of 3-SAT being solved in times like poly(n) or $2^{\mathcal{O}(\sqrt{n})}$.

At first, it's not straightforward to see whether or not SETH implies ETH, but in turns out that the answer to this question is positive, as was shown in [IP01]. Before proving that SETH implies ETH, we present two results of interest for the proof:

Since SETH deals with k-SAT and ETH with 3-SAT, we require the following reduction from k-SAT to 3-SAT. Let ψ be a k-CNF over n variables with m clauses, we show how to turn ψ into an 3-CNF ϕ over $n + m(k-3)$ variables and $m(k-2)$ clauses, such that ψ is satisfiable if and only ϕ is satisfiable.

We pick a clause C_l of ψ and we turn C_l into a 3-CNF. Say

$$C_l = (x_1 \vee x_2 \vee \ldots \vee x_k)$$

We introduce new variables $y_{1,l}, y_{2,l}, \ldots, y_{k-3,l}$, which are unique for each clause C_l. We then rewrite C_l as the CNF

$$(x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-3} \vee x_{k-1} \vee x_k)$$

It's easy to see that if a truth assignment $\alpha : \{x_1, \ldots, x_n\} \longrightarrow \{T, F\}$ satisfies C_l, we can obtain a truth assignment $\beta : \{x_1, \ldots, x_n\, y_{1,l}, \ldots, y_{k-3,l}\} \longrightarrow \{T, F\}$ that satisfies the new CNF. In particular, α must set some x_i to T, so in β we set $\beta(y_{j,l}) = T$ for all $j < i$, and $\beta(y_{j,l}) = F$ otherwise.

To obtain ϕ, we thus replace each clause C_j of ψ by the corresponding CNF. One can check that ϕ is satisfiable if and only if ψ is satisfiable. Since ϕ has $n + m(k-3)$ variables, if we assume that ETH is false, we could solve k-SAT in time $o(2^{\delta(n+m(k-3)})$ for any constant $\delta > 0$. But this doesn't contradict SETH, as there is no guarantee that $m = \mathcal{O}(n)$.

To circumvent this issue we use the Sparsification Lemma of [IP01]:

Lemma 2.1. *For all $\varepsilon > 0$ and $k \geq 3$, there is a constant $c > 0$ such that there is an algorithm that takes a k-CNF formula f on n variables, and outputs $2^{\varepsilon n}$ k-CNF formulas $f_1, f_2, \ldots f_{2^{\varepsilon n}}$ in time $\mathcal{O}(2^{\varepsilon n} \cdot \text{poly}(n))$ such that:*

- *Each f_i has n variables and $\leq c \cdot n$ clauses.*
- *f is satisfiable if and only if there is an i such that f_i is satisfiable.*

Theorem 2.1. SETH *implies* ETH*.*

Proof. Assume to the contrary that we can solve 3-SAT in $o(2^{\delta n})$ for some very small constant $\delta > 0$. Consider an instance of the k-SAT problem where we are given the k-CNF ψ over n variables and m clauses. We first apply the Sparsification Lemma to ψ with a small enough ε. We then turn each of the $2^{\varepsilon n}$ f_i into a 3-CNF f_i' using the reduction described above. Since f_i is over n variables and at most cn clauses, f_i' is over at most $n + cn(k-3)$ variables. By assumption, we can verify if f_i' is satisfiable in time $o(2^{\delta(n+cn(k-3))})$. We repeat this for all the f_i', and if any f_i' is satisfiable, we output SAT. Otherwise we output UNSAT, as by property ψ is satisfiable if and only if some f_i' is satisfiable. This procedure takes time $\mathcal{O}(2^{\delta(n+cn(k-3))} \cdot 2^{\varepsilon n} \cdot \text{poly}(n))$. By fixing k, then choosing a small ε, and finally a small enough δ, we can show that k-SAT is solvable in time $\mathcal{O}(2^{(1-\lambda)n})$ for some constant $\lambda > 0$ independent on k, which contradicts SETH. $\square$

Now, let's look at why we might believe that SETH is true. What algorithms have been proposed to solve k-SAT?

We start with a simple algorithm that only does marginally better than brute force.

Algorithm 2.1 A simple recursive algorithm for k-SAT

```
Input: A CNF $\psi$ with at most $k$ literals per clause.
if $\psi$ is empty then                                                    ▷ First base case
    Return SAT.
end if
if $\psi$ has an empty clause then                                         ▷ Second base case
    Return UNSAT.
end if
Pick a clause $C$ in $\psi$. There are at most $2^k - 1$ assignments to the variables in $C$ that make it true.
for each of these truth assignments $\alpha$ do
    Apply $\alpha$ to $\psi$ and recurse on the simplified CNF.
end for
```

Correctness follows from the behaviour of the algorithm. We thus turn our attention to the running time. Due to the recursive elimination of variables from ψ, clause C might contain fewer than k variables. However, removing exactly k variables is the worst-case. Thus, the running time is given by:

$$T(n) = (2^k - 1)T(n-k) + \text{poly}(n)$$

Solving the recurrence, we get:

$$T(n) = \mathcal{O}((2^k - 1)^{n/k} \cdot \text{poly}(n)) = \mathcal{O}([(2^k - 1)^{1/k}]^n \cdot \text{poly}(n)) = \mathcal{O}((c_k)^n)$$

Where $c_k = 2^{1-\frac{1}{\mathcal{O}(k \cdot 2^k)}}$ is a constant that depends only on k. We can however improve this result by removing the exponential factor in k from the denominator in the exponent. This leads us to another algorithm for k-SAT, Schöning's Algorithm [Sch99], running in time $\mathcal{O}(2^{(1-\frac{1}{k})n})$. While the algorithm we present is randomized, it is possible to turn it into a deterministic algorithm for k-SAT.

We first start with a subroutine:

Algorithm 2.2 A randomized algorithm for k-SAT

```
**Input:** A $k$-CNF $\psi$.
Pick a random assignment $\alpha : \{x_1, x_2, \ldots, x_n\} \longrightarrow \{T, F\}$ to the variables of $\psi$.
for $i = 1$ to poly($n$) do
    if all clauses in $\psi$ are satisfied then
        Return SAT.
    else
        Pick a random clause $C$ that is not satisfied by $\alpha$.
        Pick a random variable $x_i$ in $C$, flip the value of $\alpha(x_i)$.
    end if
end for
Return UNSAT.
```

We make the following observations about this algorithm :

- It runs in poly(n) time.
- If ψ is unsatisfiable, the algorithm always returns UNSAT.
- If ψ is satisfiable, the algorithm returns SAT, with some probability p.

As we will see next lecture, p is exponentially small. Therefore, we would like to improve the algorithm so that on any satisfiable k-CNF ψ, it outputs SAT with probability at least 2/3. To achieve this result, we can run the algorithm l times and if the algorithm outputs SAT at any of these trial, we output SAT, since we know the algorithm never ouputs SAT on an unsatisfiable ψ. We output UNSAT if for all the trials the algorithm says UNSAT. A simple probabilistic argument shows that choosing $l = \mathcal{O}(\frac{1}{p})$ guarantees that for any satisfiable k-CNF the probability of erroneously outputting UNSAT is at most 1/3.

References

[AWY15] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, 2015.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[Liu18] Sixue Liu. Chain, generalization of covering code, and deterministic algorithm for k-sat. *CoRR*, abs/1804.07901, 2018.

[Sch99] T. Schoning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 410–414, 1999.

[Wil13] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *CoRR*, abs/1312.6680, 2013.