COMS E6998-05: Fine-grained complexity (Fall'26) March 10, 2026

Lecture 8: Algorithms in other models

Instructor: *Josh Alman*

Scribes: *Collins Evans & Zev McManus (2026)*
Sayak Chakrabarti & Kevin Yeo (2022)

1 Introduction

So far in this course, we have seen some interesting problems like SETH, APSP and 3SUM, and their reductions to many other problems, giving groups of problems with correlated hardness bounds. However, until now, these groups of problems, and their corresponding conjectures have remained separate. To address this fact, we introduce alternative complexity classes and models of computation, and revisit our familiar problems in these new conditions.

2 Complexity Classes

- **TIME(T):** This is the standard complexity class we have worked in throughout the course. Given an instance x and a language L, if we can decide in deterministic time T whether $x \in L$ or not, then we say that $L \in \mathsf{TIME}(T)$.
- **NTIME(T):** Given an instance x and a language L, let us assume that a prover gives us a proof w. We say $L \in \mathsf{NTIME}(T)$ if we have deterministic time to check x, w and infer:
  - $x \in L$ if w is the proof that x is an accept instance.
  - $x \notin L$ if we reject all proofs.
- **Co-NTIME(T):** Given an instance x and a language L, let us assume that a prover gives us a proof w. We say $L \in \mathsf{NTIME}(T)$ if we have deterministic time to check x, w and infer:
  - $x \in L$ if no such proof exists.
  - $x \notin L$ if there is a proof w that makes us reject.

From our knowledge of this course so far, we can infer the following about $k - \mathsf{SAT}$:

- $k - \mathsf{SAT} \in \mathsf{TIME}(2^{n\left(1-\frac{1}{O(k)}\right)})$.
- $k - \mathsf{SAT} \in \mathsf{NTIME}(\mathsf{poly}(n))$ (w is an assignment which causes the k-CNF to evaluate to true).
- $k - \mathsf{SAT} \in \mathsf{Co\text{-}NTIME}(2^{n\left(1-\frac{1}{O(k)}\right)})$ (Ignore the proof and use our deterministic SAT algorithm directly; no better approach is currently known).

Based on this, there is another (less believable conjecture) called the NSETH.

Conjecture 1 (NSETH). *For every $\epsilon > 0$, there exists some $k > 2$ such that $k-\mathsf{SAT} \notin$ Co-NTIME$(2^{(1-\epsilon)n})$.*

Similarly, we have the language 3SUM in the complexity class as:

- 3SUM $\in \mathsf{TIME}(n^2/\log n)$.
- 3SUM $\in \mathsf{NTIME}(n)$ (We are given with the three numbers).
- 3SUM $\in$ Co-NTIME$(n^{1.5}\log^2 n)$ (A not-so-trivial proof which will be shown later).

APSP is not a decision problem, but we can pick our favorite subcubic equivalent problem to categorize into the complexity classes. For example, using Exact Triangle:

- Exact Triangle $\in \mathsf{TIME}(n^3/2^{\sqrt{\log n}})$.
- Exact Triangle $\in \mathsf{NTIME}(n)$.
- Exact Triangle $\in$ Co-NTIME$(n^{2.9})$ (Using a similar proof as that of Theorem 2)

3 Barriers to Proofs

We would like to prove something like "SETH implies the 3SUM Conjecture" so that we don't need multiple independent conjectures. Unfortunately, we do not know how to unify these conjectures, and co-nondeterminism gives barriers to potential proofs.

Theorem 1. *Assuming* NSETH*, there is no deterministic fine-grained reduction showing that* SETH *implies* 3SUM *conjecture.*

Theorem 1 can be inferred as an algorithm for $k-\mathsf{SAT}$ that has access to an oracle for 3SUM, such that if the black box solves instances of size m in time $O(m^{2-\epsilon})$, then the algorithm solves $k-\mathsf{SAT}$ in time $O(2^{n(1-\delta)})$, for some $\delta > 0$ depending on ϵ.

Proof of Theorem 1. For the sake of contradiction, let us assume that we have the above algorithm of the reduction. Our goal now is to make a co-nondeterministic algorithm $k-\mathsf{SAT}$. Whenever we make a blackbox call to 3SUM, if it returns YES, then we return the non-deterministic proof of this, otherwise the co-nondeterministic proof. This will solve $k-\mathsf{SAT}$ in time Co-NTIME$(2^{(1-\delta)n})$, refuting NSETH. □

Theorem 2. 3SUM $\in$ Co-NTIME$(n^{1.5}\log^2 n)$

Proof. The proof consists of the three steps:

- Choose a prime $p \in [n^{1.5}, n^{1.5}\log n]$,
- Guess a number $t \leq n^{1.5}\log n$ along with t triplets $(i,j,k),\ i,j,k \in [n]$ such that the sum $a_i+b_j+c_k \equiv 0 \bmod p$ but not 0 over $\mathbb{Z}$.
- Verify that for all $\{(i_1,j_1,k_1),\ldots,(i_t,j_t,k_t)\}$, $a_{i_\ell}+b_{j_\ell}+c_{k_\ell} \equiv 0 \bmod p$ but $a_{i_\ell}+b_{j_\ell}+c_{k_\ell} \neq 0$.

Lemma 3. *A proof w always exists for a NO instance.*

Lemma 3 implies that we can always find a prime in the given range such that t is not too large, i.e. $\leq n^{1.5}\log n$. Now, this proof can also be easily verified from the next lemma

Lemma 4. *We can verify the proof w given as t triplets in $O(n^{1.5}\log^2 n)$ time.*

From these two lemmas, we can solve 3SUM in Co-NTIME($n^{1.5}\log^2 n$). □

Proof of Lemma 3. The number of primes in the given range is $\Omega(n^{1.5})$. Now, each triplet $a_i + b_j + c_k \equiv 0 \bmod p$ for at most $O(1)$ of those primes. This is true as the total sum is at most n^c, and since the primes are greater than $n^{1.5}$, at most $c/1.5$ many primes in the range can divide it. Thus, the average value of t over these primes is less than $\frac{n^3.O(1)}{n^{1.5}} = O(n^{1.5})$, and a t which is less than this average can be chosen. Using the corresponding prime p completes the proof. □

Proof of Lemma 4. We check all the triplets (i_ℓ, j_ℓ, k_ℓ) and verify that they are solutions modulo p but not over $\mathbb{Z}$, which will take time $O(t + p\log p) = O(n^{1.5}\log^2 n)$. Alongside the verification, we can also count the number of such triplets and verify whether that is equal to t. □

4 Linear Decision Trees

When working with real numbers we cannot use the standard Word-RAM model (which can store w-bit words) and must pick a suitable model of computation. One common choice is the Real-RAM model where a word can be any real number, but this often leads to algorithms which act "unreasonably" (for example, packing an n-bit input into a single value). Linear Decision Trees provide an alternative model for working with real numbers.

Definition 5. ***Linear Decision Tree (LDT)****: A binary tree where each node is associated with a linear inequality (for example, $x_4 + x_2 - 2x_4 + 100\pi x_{12} \geq 0$) and each leaf is labeled with a possible output.*

We compute using an LDT as follows:

- Start at the root.
- Evaluate the current node's linear inequality.
- If the inequality evaluates to true, go to the left child. Otherwise, go to the right child.
- Once a leaf is reached, output its corresponding label.

See Figure 1 for a visual representation.
A few key properties of LDTs:

- **Width** w: Each inequality uses $\leq w$ variables, taking constant (or perhaps logarithmic) time to evaluate.
- **Running time**: The running time of the algorithm is equivalent to the depth of the LDT.
- **Nonuniform model**: We must use a different LDT for each input size n (i.e., we need to know the input size to build the tree). This is a common property in circuit models.

Fact 6. *Sorting a list of n numbers can be done with an LDT of depth $O(n\log n)$ (use any common comparison-based sorting algorithm).*

Theorem 7. *The $(\min, +)$ matrix product has an LDT of depth $O(n^{2.5}\log^2 n)$.*

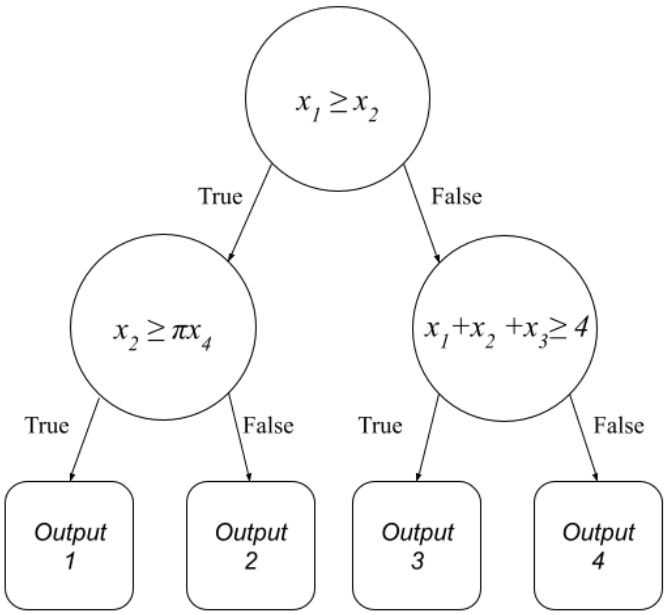

Figure 1: A sketch of a Linear Decision Tree.

The high-level idea is to narrow down the answer to one possibility without initially knowing what that possibility actually is. To prove this theorem, we first introduce the following lemma:

Lemma 8. *The* $(\min, +)$ *product of an* $n \times d$ *matrix and a* $d \times n$ *matrix has an LDT of depth* $O(nd^2 \log n)$.

Proof of Theorem assuming Lemma 8. To compute the $(\min, +)$ product of an $n \times n$ matrix, it suffices to compute n/d instances of the $(\min, +)$ product of $n \times d$ and $d \times n$ matrices, and combine the results. The total depth is bounded by:

$$O\left(nd^2 \log n \cdot \frac{n}{d} + n^2 \cdot \frac{n}{d}\right) = O\left(n^2 d \log n + \frac{n^3}{d}\right)$$

Setting $d = \sqrt{n}$ gives an overall LDT depth of $O(n^{2.5} \log^2 n)$. □

Proof of Lemma 8. Given matrices $A, B \in \mathbb{R}^{n \times d}$, we want to compute $C[i, j] = \min_{k=1}^{d}(A[i, k] + B[k, j])$. Equivalently, for every $i, j \in [n]$, we want to find k^* such that for all $k \in [d]$:

$$A[i, k^*] + B[k^*, j] \leq A[i, k] + B[k, j]$$

Using "Fredman's Trick", we can rewrite this inequality as:

$$A[i, k^*] - A[i, k] \leq B[k, j] - B[k^*, j]$$

We construct the LDT as follows:

1. Initialize L as an empty list.

2. For all $i \in [n]$ and all $k, k' \in [d]$, add $a_{i,k,k'} = A[i, k'] - A[i, k]$ to L. Similarly, add $b_{j,k,k'} = B[k, j] - B[k', j]$ to L.

3. Sort L. This requires an LDT of depth $\tilde{O}(nd^2)$. The specific leaf we reach in this tree determines the sorted order of the elements.

4. For all $i, j \in [n]$, consider the sublist of L containing just $a_{i,k,k'}$ and $b_{j,k,k'}$. The sorted order of these elements completely determines the k^* in Fredman's trick. Our LDT can now output that k^*.

□

Note on step 4: In a standard Word-RAM model, scanning the sorted list to find k^* for every i, j would require additional time. However, because the LDT is a non-uniform model, the specific leaf reached after sorting uniquely identifies the permutation of L. The correct k^* for every i, j pair is therefore pre-computed and hardcoded directly into the output of that leaf, meaning it takes zero additional depth (time) to produce the final answer.

4.1 Other Results

We just demonstrated a width-4 LDT of depth $\tilde{O}(n^{2.5})$ for APSP. Further research has shown the following:

- APSP: [KLM19] showed that if we allow a width-8 LDT, we can achieve a depth of $\tilde{O}(n^2)$ for APSP (which matches the trivial lower bound of looking at all inputs).
- 3SUM:
  - [Eri99] showed that width-3 LDTs for 3SUM require depth $\Omega(n^2)$.
  - [GP14] provided a width-4 LDT of depth $\tilde{O}(n^{1.5})$.
  - [KLM19] provided a width-6 LDT of depth $\tilde{O}(n)$.

Even though the LDT is a heavily theoretical model, researchers have successfully used these ideas to develop faster algorithms in standard computational models. In fact, the first algorithms to shave off polylogarithmic factors for 3SUM and APSP were derived directly from LDTs.

References

[Eri99] Jeff Erickson. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 1999, 1999.

[GP14] Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 621–630. IEEE, 2014.

[KLM19] Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-sum and related problems. *Journal of the ACM (JACM)*, 66(3):1–27, 2019.