COMS E6998-15: Fine-grained complexity (Fall'22) February 10, 2026

Lecture 4: APSP & Subcubic Reductions

Instructor: *Josh Alman*

Scribes: *Hideaki Takahashi (2026)*
Rashida Hakim, Hantao Yu (2022)

1 Introduction

Most of the results we have talked about previously in this course have been about proving there are no subquadratic time algorithms for certain problems (OV, longest common substring, etc.). In this lecture, we discuss reductions to and from the all-pairs shortest path problem (APSP), where the fastest known algorithms run in approximately cubic time.

APSP. Given a directed, weighted graph G with integer weights in the set $\{1, 2, ..., n^c\}$ compute the shortest path between each of the n^2 pairs of vertices.

Since the output has size $\Omega(n^2)$, the runtime for any APSP algorithm has to be at least $\Omega(n^2)$. One classic algorithm for this problem is the Floyd-Warshall algorithm, which is a dynamic programming algorithm which runs in $O(n^3)$ time. The current fastest algorithm for APSP runs in time $O\left(n^3/2^{\Omega(\log(n))^{1/2}}\right)$ [Wil14]. This is faster than any polylog speedup (of n^3) and slower than any polynomial speedup.

APSP Conjecture No polynomial speed-up of current APSP algorithms is possible. Formally, there exists a large enough c such that APSP with weights $\{1, 2, ..., n^c\}$ cannot be solved in time $O(n^{3-\epsilon})$ for any $\epsilon > 0$.

There are modifications of APSP that have truly subcubic algorithms. On unweighted and undirected graphs, Seidel's algorithm [Sei95] solves APSP in time $O(n^\omega \log(n))$. For weighted, undirected graphs, the Shoshan-Zwick algorithm [SZ99] solves APSP in time $O(n^{\omega+c})$, which can be truly subcubic if c (where n^c bounds the graph edge-weights) is small. Note that $\omega < 3$ is the matrix multiplication exponent (details below).

2 Matrix Multiplication

Matrix Multiplication (MM) Given two n by n matrices A and B where the entries of A and B are from the set $\{1, 2, ..., n^c\}$, we are tasked with computing an n by n matrix $C = A \times B$ defined as follows:

$$C[i, j] = \sum_{k=1}^{n} A[i, k]B[k, j]$$

We now define ω as the infimum over the set of numbers t such that an algorithm running in time $O(n^{t+o(1)})$ can perform matrix multiplication (MM). It is easy to show that $2 \leq \omega \leq 3$. The naive algorithm for MM runs in $O(n^3)$ since it performs n multiplications to compute each one of the n^2 entries in C. On the other hand, simply reading all the entries of A and B takes $\Omega(n^2)$ time so that establishes a lower bound on ω.

Strassen's Algorithm The earliest improvement to the naive MM algorithm is Strassen's algorithm in 1969 [Str69], which shows that $\omega \leq 2.81$. The naive algorithm for MM on 2 by 2 matrices requires $2^3 = 8$ multiplications, but Strassen's approach reduces this number to 7. This same approach can be applied to larger matrices by treating them as 2 by 2 block matrices.

The algorithm defines the following 7 matrices (each requiring one multiplication to compute):

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22})B_{11} \\
M_3 &= A_{11}(B_{12} - B_{22}) \\
M_4 &= A_{22}(B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12})B_{22} \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}$$

Then these matrices can be used to compute $C = A \times B$ without any additional multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

The current fastest algorithms for MM show that $\omega \leq 2.373$ [AW21].

(min, +) Matrix Multiplication We define a variant of MM called $(\min, +)$-MM as follows: We are given two n by n matrices A and B where the entries of A and B are in the set $\{1, 2, ..., n^c\}$. The goal is to compute an n by n matrix $C = A \star B$ such that:

$$C[i, j] = \min_k (A[i, k] + B[k, j])$$

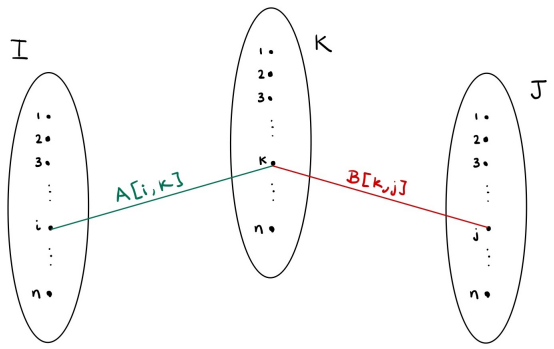
We might initially imagine that we can do some Strassen-like approach to improve the naive algorithm (which runs in $O(n^3)$) for $(\min, +)$-MM. However, note that Strassen's algorithm relies on using subtraction, which is the inverse of the addition operation done by the summation in the definition of MM. In $(\min, +)$-MM, we have replaced the summation with a minimum, which does not have an inverse. So a direct translation of Strassen cannot be used to solve this problem, and in fact, we will show that the APSP Conjecture implies no polynomial speedup is possible for $(\min, +)$-MM.

Theorem 1. *There is a $\epsilon > 0$ such that $(\min, +)$-MM can be computed in time $\mathcal{O}(n^{3-\epsilon})$ if and only if there is $\delta > 0$ such that APSP can be computed in $\mathcal{O}(n^{3-\delta})$.*

The proof of this theorem follows from the following two lemmas.

Lemma 2. *If* APSP *can be solved in time* $T(n)$ *then* $(\min, +)$-MM *can be solved in time* $O(T(n))$.

Proof. Given an instance A, B of $(\min, +)$-MM, create a graph G as follows: Create 3 sets of n vertices I, K and J. Then, for all $i \in I, j \in J$, add the edge (i, k) with weight $A[i, k]$. Next, for all $k \in K, j \in J$ add the edge (k, j) with weight $B[k, j]$.



G has the property that $d(i, j) = C[i, j]$ where $C = A \star B$. This is because each path from i to j must pass through some vertex k and therefore has length $A[i, k] + B[k, j]$. Hence, the shortest path between i and j has length $\min_k(A[i, k] + B[k, j])$, which matches the definition of $C[i, j]$. Thus, if there existed an algorithm for APSP that took $T(n)$, then applying this algorithm to G would allow $(\min, +)$-MM to be solved in $O(T(n))$. □

Lemma 3. *If* $(\min, +)$-MM *can be solved in time* $T(n)$ *then* APSP *can be solved in time* $O(T(n)\log(n))$.

Proof. Given graph G, create a complete graph G' from G by adding an edge between any disconnected pair of vertices that has weight $n^{c+1} + 1$ (this large weight ensures it will not be used in any shortest path). Then let A be the weighted adjacency matrix representing G', with 0 on the diagonal entries.

$$A[i, j] = \begin{cases} 0 & i = j \\ w(i, j) & \end{cases}$$

For positive integer q, let A_q be the (min, +) product of q copies of A.

$$A_q = \underbrace{A \star A \star ... \star A}_{q}$$

We now claim that $A_q[i, j]$ is the smallest weight path from i to j that uses $\leq q$ edges. We can prove this claim using induction. The base case of $q = 1$ is true by the definition of A. Assume that the claim is true for A_q. $A_{q+1} = A_q \star A$. So $A_{q+1}[i, j] = \min_k(A_q[i, k] + A[k, j])$. $A_{q+1}[i, j]$ is the minimum weight path with $\leq q + 1$ edges from i to j. The first term of the expression is the minimum weight path with $\leq q$ edges from i to k (inductive assumption) and the second term is the weight of the additional edge from k to j. If $k = j$, the expression represents the minimum weight path with $\leq q$ edges from i to

j, since $A[j,j] = 0$. So minimizing over all k gives the minimum weight path with $\leq q+1$ edges as desired.

The shortest path between any two vertices in the graph requires at most n edges (if more edges were used, pigeonhole principle implies some vertex was repeated, which is redundant for a shortest path in a graph with non-negative weights). So to solve APSP, we need to calculate A_q for $q \geq n$. We can do this via repeated squaring with the $(\min, +)$-MM operator.

$$A_1 = A, A_2 = A \star A, A_4 = A_2 \star A_2, ...$$

This implies that we can solve APSP using $O(\log_2(n))$ applications of the $(\min, +)$-MM operator. Thus, if $(\min, +)$-MM can be solved in time $T(n)$ then APSP can be solved in $O(T(n)\log(n))$. □

3 Triangles

Triangle Detection Given an unweighted, undirected graph G with n nodes, does G have a triangle? The trivial algorithm, that goes over all possible triplets and checks for a triangle, takes $O(n^3)$ time, and in fact one can solve it in $O(n^\omega)$ time: let A be the adjacency matrix of G. Then we know

$$A^q[i,j] = \text{ number of paths of length exactly } q \text{ from } i \text{ to } j.$$

Check if $A^3[i,i] > 0$ for some $i \in [n]$. If so, output true, else output false. Note that A^3 can be computed using two invocations of MM, hence the $O(n^\omega)$ running time.

Min-Weight Triangle Given an unweighted, undirected graph G with n nodes, find the triangle with minimum sum of weights (weights of edges lie in $\{-n^c, -n^c+1, \ldots, n^c\}$). We can reduce the Min-weight Triangle problem to $(\min, +)$-MM and give an algorithm with running time $O(n^3/2^{\Omega(\log n)^{1/2}})$ as: let A be the weighted adjacency matrix and $A_2 = A \star A$, and return

$$\min_{i \neq j} A_2[i,j] + w(i,j).$$

This is correct because recall that

$$A_2[i,j] = \min_k (A[i,k] + b[k,j]) = \min_k (w(i,k) + w(k,j)).$$

Every triangle (i,j,k) in G can be found in this way as we first choose different i,j and then choose k. The formula above minimizes the sum of these three weights.

Negative Triangle Given a weighted, undirected graph G with n nodes, find a triangle (i,j,k) whose sum of weights is negative.

Theorem 4. *If* Negative Triangle *can be solved in time* $O(n^{3-\varepsilon})$*, then* APSP *with weights in* $\{-W, \ldots, W\}$ *can be solved in time* $O(n^{3-\varepsilon/3}\log(W))$.

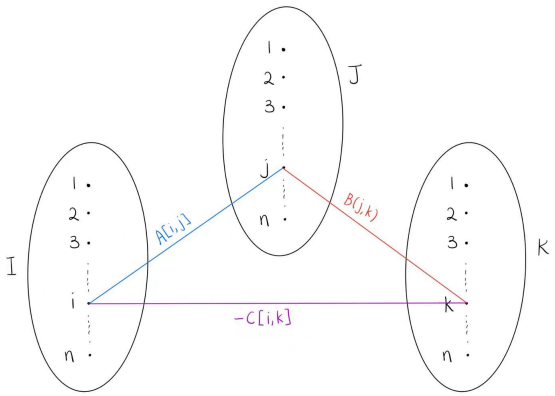
To prove this theorem, we need another intermediate problem:

All-Pairs Negative Triangle (APNT) Given a tripartite undirected weighted graph G, $V = I \cup J \cup K, E \subset I \times J \cup J \times K \cup I \times K$, for all $i \in I, k \in K$, return whether there is a $j \in J$ such that (i, j, k) is a negative triangle.

Proof of Theorem 4. Since we already showed that APSP and $(\min, +)$-MM can be reduced to each other, we will reduce as follows:

$$\text{APSP} \to (\min, +)\text{-MM} \to \text{APNT} \to \text{Negative Triangle}.$$

We first reduce $(\min, +)$-MM to APNT. we are given two matrices A, B. Now pick a matrix C and construct the following graph:



Solve APNT on this graph, and observe that $i \in I, k \in K$ are a part of a negative triangle iff $(A \star B)[i, k] < C[i, k]$. This is because $(A \star B)[i, k] - C[i, k] < 0$ iff there exists a $j \in J$ such that $A[i, j] + B[j, k] - C[i, k] < 0$ iff (i, k) is a part of a negative triangle.

Now do a simultaneous binary search. Here "simultaneous" means we are doing a binary search for each (i, k) pair. Initially set $C[i, k] = W$ for all i, k. Every time we run APNT, we know whether (i, k) is a part of a negative triangle, which is equivalent to knowing whether $(A \star B)[i, k] < C[i, k]$ by our previous argument. Therefore, we need to run APNT $\log(2n^c) = c \log 2n$ times to find out $(A \star B)[i, k]$ for all (i, k).

The running time for our algorithm is $\log(2W)$ times the running time of APNT algorithm plus the time to construct the graph, which is $O(n^2)$.

Secondly, we reduce APNT to Negative Triangle. Given an algorithm NT for Negative Triangle, we proceed as follows:

Correctness: Since we partition I, J, K into I', J', K', if any negative triangle (i, j, k) exists, it must exist in one of the I', J', K' so that we will eventually find it. In addition, each time we fine a negative triangle for (i, k), we set $w(i, k) = \infty$ to make sure that future NT algorithm would not consider (i, k) again and return any (i, j', k) as this triangle could not possibly have negative weight.

Running time: let $T(n)$ be the running time for NT on n nodes, then the running time of the the above algorithm is then

$$T(s) \cdot \left[\left(\frac{n}{s}\right)^3 + n^2\right] \leq s^{3-\varepsilon} \cdot \left[\left(\frac{n}{s}\right)^3 + n^2\right] = n^{3-\varepsilon/3},$$

assuming $T(n) \leq O(n^{3-\varepsilon})$.

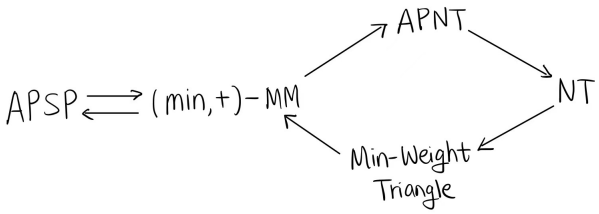
Algorithm 1: REDUCTION ALGORITHM(G)

Result: For all $i \in I, k \in K$, return whether there is $j \in J$ such that (i, j, k) is a negative triangle
Input: Tripartite undirected weighted graph $G = (V, E) = (I \cup J \cup K, E)$
Algorithm:

```
Partition each of I, J, K into n/s groups of sets I', J', K' of size s
for I', J', K' do
    while NT(I', J', K') ≠ false do
        NT(I', J', K') = (i, j, k)
        return true for (i, k)
        set w(i, k) = ∞
    end
end
```

In all, if we have an algorithm for Negative Triangle running in time $T(n) \leq O(n^{3-\varepsilon})$, we get an algorithm for APNT with running time $O(n^{3-\varepsilon})$, which further gives us an algorithm for $(\min, +)$-MM with running time $O(n^{3-\varepsilon} \log W)$. □

Summary In conclusion, today we showed



We can do most of the reductions for subcubic problems in both directions, like we did in this lecture (summarized by the picture above). However, recall that the subquadratic ones mostly only work in one direction. For example, SETH implies OVC and ETH but we do not know if the converse is true or not. Another example is that CVC (Correlated Vector Conjecture) implies OVC but the converse is again not necessarily true.

References

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences*, 51(3):400–403, 1995.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

[SZ99] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 605–614, 1999.

[Wil14] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC*, pages 664–673. ACM, 2014.