Suppose we want to find a triangle in a graph, i.e. given a graph on n nodes, find vertices u, v, w such that edges $(u, v), (v, w)$, and (u, w) are in the graph. This can be solved in $O(n^\omega)$ time by cubing the adjacency matrix. In fact, if A is the adjacency matrix, then $tr(A^3) = 6 \cdot$ (number of triangles), where tr denotes the trace of the matrix.

But what if we want to find triangles of *minimum weight*? We will address both the node-weighted and edge-weighted versions.

1 Finding triangles with minimum node weights

In this problem, the vertices of our graph have node weights $w : V \to \mathbb{Z}$, and we want to find a triangle such that the sum of its node weights is minimized. If the weights are integers at most W, this problem can be solved in $O(W^3 n^\omega)$ simply by trying all combinations of triples of weights, however this is a bad dependence on W. In fact, the problem can be solved in time $\tilde{O}(n^\omega)$. First we will give a warm-up algorithm which runs in time $O(n^{2.529})$.

1.1 Warm-up: an $O(n^{2.529})$ time algorithm

We will the so called *min-witness product* of Boolean matrices. For Boolean matrices A and B their min-witness product $A \circ B$ is defined as

$$(A \circ B)[i, j] = \min\{k | A[i, k] = B[k, j] = 1\}.$$

Using this product, we can find minimum-weighted triangles as follows:

Algorithm 1: Finding triangles with minimum node weights

(1) Make three copies of the graph G; call them I, J, K. If $(u, v) \in E$, draw an edge between $u \in I$ and $v \in K$, and between $u \in K$ and $v \in J$.
(2) Sort the nodes in K by their weight.
(3) Compute the min-witness product $B \circ B$, where B is the adjacency matrix of the new graph.
(4) Compute $\min_{(i,j)\in E} w(i) + w(j) + w((B \circ B)[i, j])$.

Below we will show that the min-witness product of two $n \times n$ matrices can be computed in $O(n^{2.529})$ time. Therefore, minimum node weight triangles can be found in $O(n^{2.529})$ time.

Lemma 1.1. *The min-witness product of two $n \times n$ Boolean matrices can be computed in $O(n^{2.529})$ time.*

Proof. We want to compute the min-witness product C of A and B.

Let P be a parameter. Split the columns of A and corresponding rows of B into n/P buckets of P consecutive elements. Denote the buckets by $T_1, \ldots, T_{n/P}$, where $T_i = \{iP + 1, \ldots, (i + 1)P\}$ for each i.

Let A_i be the submatrix of A restricted to the columns in T_i, and let B_i be the submatrix of B restricted to the rows in T_i. For every i, multiply $A_i \cdot B_i$ (BMM) to obtain C_i. This takes overall time $O((n/P)M(n, P, n))$ where $M(n, p, n)$ is the running time to multiply an $n \times p$ by a $p \times n$ matrix.

Notice that for every j, k, $C_i[j, k] = 1$ if and only if there is some $t \in T_i$ with $A[j, t] = B[t, j] = 1$. In $O(n^3/p)$ time, we can determine for every $j, k \in [n]$ the smallest i for which $C_i[j, k] = 1$; denote this by $i(j, k)$. Notice that the min-witness product entry $C[j, k]$ is contained in $T_{i(j,k)}$.

For every $j, k \in [n]$, try all P elements of $T_{i(j,k)}$ to find the minimum $t \in T_{i(j,k)}$ such that $A[i, t] = B[t, j] = 1$. This is $C[j, k]$.

The total running time is

$$O(n^2P + nM(n, P, n)/P),$$

and is minimized for P such that $P^2n = M(n, P, n)$. For the current best bounds on rectangular matrix multiplication, we get that $P = n^{0.529}$ minimizes the expression, and we get an $O(n^{2.529})$ time algorithm.

If $\omega = 2$, we'd get $O(n^{2.5})$. □

1.2 An $n^{\omega+o(1)}$ time algorithm for min-node-weight triangle

Czumaj and Lingas proved that the min-node-weight triangle problem can be solved in $n^{\omega+o(1)}$ time. We will present their algorithm here.

Algorithm overview The algorithm works as follows. First we make G tripartite as in the warm-up, so that the vertices have three parts I, J, K. Then, we sort the vertices of I, J and K by weight. Then, we split each of I, J, K into g consecutive chunks (with respect to the sorted order) of n/g vertices each. Then we recurse. If we recurse on all triples of chunks of I, J, K, the running time will be too large. Instead, we will show that we can recurse on only $O(g^2)$ triples of chunks.

The full algorithm is given in Algorithm 2.

Algorithm 2: Finding triangles with minimum edge weights

(1) Make three copies of the nodes of graph G; call them I, J, K. If $(u, v) \in E$, draw an edge between $u \in I$ and $v \in K$, between $u \in K$ and $v \in J$ and between $u \in I$ and $v \in J$.
(2) Sort the nodes in each of I, J, and K by their weight.
(3) Split each of I, J, and K into g consecutive chunks (with respect to the sorted order) of size n/g each.
(4) For every triple of chunks, one from each of I, J, and K, run a triangle detection algorithm on the graph induced by the union of the chunks. This takes $O(g^3(n/g)^\omega) = O(n^\omega g^{3-\omega})$ time. Now we have a list of at most g^3 triples containing triangles.
(5) Remove the *dominated triples*: If triples (i, j, k) and (i', j', k') both have triangles and $i < i'$, $j < j'$, and $k < k'$, then we say that (i, j, k) dominates (i', j', k'). In this case, note that we don't need to recurse on (i', j', k') since the triangles in (i, j, k) are at least as good as those in (i', j', k'). Go over all $O(g^6)$ pairs of triples returned by the previous step and remove all dominated ones.
(6) Recurse on the remaining (undominated) triples. (In the base case, if $n = g$, run an $O(g^3)$ time brute force algorithm.)

Running time analysis The running time before the recursion is $O(g^{3-\omega}n^\omega)$ (for the triangle detection in each triple) and $O(g^6)$ to remove all triples that are dominated. Then one recurses on a set of undominated triples each of size n/g.

To bound the running time, we will bound the number of undominated triples.

Lemma 1.2. *Let $S \subseteq [g]^3$. If no triple in S is dominated by another triple in S, then the size of S is at most $3g^2$.*

Proof. We let $(i, j, k) < (i', j', k')$ denote that (i, j, k) dominates (i', j', k'). Consider for all $g_1, g_2 \in [g]$ the following chains of triples from $[g]^3$:

$$(1, g_1, g_2) < (2, g_1 + 1, g_2 + 1) < (3, g_1 + 2, g_2 + 2) < \dots$$
$$(g_1, 1, g_2) < (g_1 + 1, 2, g_2 + 1) < (g_1 + 2, 3, g_2 + 2) < \dots$$
$$(g_1, g_2, 1) < (g_1 + 1, g_2 + 1, 2) < (g_1 + 2, g_2 + 2, 3) < \dots$$

There are $3g^2$ such chains. Note that every triple in $[g]^3$ is in some chain, and so every triple of S is in some chain. However, S can have at most 1 triple in each chain since S contains no dominated triples. This means that any set S of undominated triples can contain at most $3g^2$ triples. □

To complete the running time analysis we solve the recurrence that arises from the recursive procedure:

$$\begin{aligned} T(n) &\leq (3g^2)T(n/g) + g^{3-\omega}n^\omega + g^6 \\ &\leq \sum_{i=0}^{\log_g n} 3^i(g^{2i})(g^{3-\omega}(n/g^i)^\omega + g^6) \\ &\leq \sum_{i=0}^{\log_g n} 3^i(n^\omega g^{3-\omega} + g^6) \qquad \text{since } \omega \geq 2 \\ &\leq n^{\omega+o(1)} \qquad \text{setting } g = \frac{\log n}{\log\log n} \end{aligned}$$

2 Finding triangles with minimum edge weights

Suppose we have a graph with *edge* weights $w : E \to \mathbb{Z}$. This time the problem is to find vertices i, j, k minimizing $w(i, j) + w(j, k) + w(i, k)$. There is **n**o known $O(n^{3-\epsilon})$ time algorithm for this (when $\epsilon > 0$). However, we can trivially solve this in $O(n^3)$ time by trying all triplets of vertices.

In this section we will show that this problem is subcubically equivalent to APSP. That is, the min-edge-weight triangle problem admits a truly subcubic time algorithm if and only if APSP admits a truly subcubic time algorithm.

First we show the easy direction:

Theorem 2.1. *If for some $\epsilon > 0$, APSP is in $O(n^{3-\epsilon})$ time then min-edge-weight triangle is in $O(n^{3-\epsilon})$ time.*

Proof. We will show that we can reduce the min-edge-weight triangle problem to the $(\min, +)$ product. Showing this completes the proof because we know that APSP and $(\min, +)$ product are equivalent. Let A be the weighted adjacency matrix (where $A[i, j] = w(i, j)$ if $(i, j) \in E$, and ∞ otherwise). We can compute the (min, +) product $(A \star A)$, where $(A \star A)[i, j] = \min_k A[i, k] + A[k, j]$. Then the minimum weight of a triangle is $\min_{(i,j)\in E} w(i, j) + (A \star A)[i, j]$. □

Now we will show the hard direction:

Theorem 2.2. *If for some $\epsilon > 0$, min-edge-weight triangles can be found in $O(n^{3-\epsilon})$ time, then APSP is in $\tilde{O}(n^{3-\epsilon/3})$ time.*

We will prove this theorem through the following string of reductions:

APSP $\to$ $(min, +)$-product $\to$ all pairs negative triangles $\to$ negative triangle $\to$ min-edge-weight triangle

All pairs negative triangles (APNT) is defined as follows. Given a tripartite graph G as before, determine for all $u_I \in I$ and $t_K \in K$ whether there exists a $v_J \in J$ such that $w(u_I, v_J) + w(v_J, t_K) + w(u_I, t_K) < 0$.

We have already shown APSP $\to$ $(min, +)$-product. Also, negative triangle $\to$ min-edge-weight triangle is immediate.

2.1 Preliminaries

Without loss of generality, we can assume that

1. For all vertices i, j, we have $(i, j) \in E$. This is because suppose that the edge weights are in $\{-M, \ldots, M\}$, where $M \geq 1$ is an integer. Then if $(i, j) \notin E$, we can add (i, j) to E with weight $w(i, j) = 6M$. This would mean that if the non-edge is part of a triangle, then its weight is greater than that of any real triangle.

2. G is tripartite. Create a graph containing three copies of G, and call them I, J, K. Draw edges between $u \in I$ and $v \in J$ with weight $w(u, v)$, and do the same thing for the other pairs of graph copies. Now our problem is equivalent to finding $u_I \in I$, $v_J \in J$, and $t_K \in K$ such that $w(u_I, v_J)+w(v_J, t_K)+w(u_I, t_K)$ is minimized.

2.2 Reducing $(min, +)$-product to APNT

Proof. We will use the following standard graph interpretation of $(min, +)$-product. We have a tripartite graph I, J, K, for each $i \in I$, $k \in K$, there is an edge of weight $A(i, k)$ from i to k, and for each, $k \in K$, $j \in J$ there is an edge of weight $B(k, j)$ from k to j. Now, the goal is to compute the minimum weight 2-path from each $i \in I$ to each $j \in J$. Suppose all weights are in $[1, n^c]$.

For all $u_I \in I, t_J \in J$, we can use binary search to guess the value $W_{ut} = \min_{v_K \in K} w(u_I, v_K)+w(v_K, t_J)$. For each u, t, we guess a value W_{ut}, and replace the edge weight $w(u_I, t_J)$ in the graph with W_{ut}. Then we can use the negative triangle algorithm to ask for each u, t, if there exists a v_J such that $w(u_I, v_J)+w(v_J, t_K) < -W_{ut}$. This would tell us if $\min_{v_J} w(u_I, v_J) + w(v_J, t_K) < -W_{ut}$. To do this, we use a simultaneous binary search (for all u, t) over all possible edge weights. This takes $O(T(n) \log M)$ time. □

2.3 Reducing APNT to NT

This is the same proof as the reduction from BMM to triangle finding that we have seen in a previous lecture.

Suppose we have an algorithm A that detects a negative triangle in $T(n)$ time. Pick a parameter L, and partition each of I, J, K into n/L pieces of size L.

Algorithm 3: All-pairs negative triangles (given an algorithm for negative triangle)

```
Partition I, J, K.
Initialize C to a matrix of all zeros. (At the end of the algorithm, C[i, j] = 1 iff (i, j) is used in a negative
 triangle.)
for all triplets (i, j, k), where i, j, k range from 1 to n/L do
    Consider G_ijk, the subgraph induced on I_i ∪ J_j ∪ K_k.
    while G_ijk contains a negative triangle do
        Let a_I, b_J, c_K be the vertices of the negative triangle.
        Set C[a, c] = 1.
        Delete (a_I, c_K) from G (this deletes it from all the induced subgraphs G_ijk).
return C
```

Running time This algorithm runs in time

$$T(L)\left(n^2 + \left(\frac{n}{L}\right)^3\right).$$

The n^2 comes from the while loop. The while loop can only run at most n^2 times because C only has n^2 elements, and on each iteration we're setting one of them to 1 (and removing the edge so we can't set it to 1 again). Each time the while loop runs, it costs $T(L)$ time to find the negative triangle.

The $(n/L)^3$ term is there because we need to check for triangles for each triplet (i, j, k), to make sure G_{ijk} has no more negative triangles.

To minimize the runtime, we set $L = n^{1/3}$, which gives a runtime of $O(n^2 T(n^{1/3}))$. Since $T(n) = n^{3-\epsilon}$, the runtime is $O(n^{3-\epsilon/3})$.

3 Application to graph radius

In the **graph radius** problem, we are given an undirected graph with integer edge weights, and want to find

$$\min_v \max_u d(u, v).$$

We may want to find the "center" vertex c such that the maximum distance from c to the rest of the graph is minimized. The graph radius is used a lot in social network analysis.

The only known algorithm for computing the radius of a graph is to solve APSP. We will show that in fact the radius problem is subcubically equivalent to APSP.

Theorem 3.1. *If for some $\epsilon > 0$, radius is in $O(n^{3-\epsilon})$ time then APSP is in $O(n^{3-\epsilon})$ time.*

Proof. We will reduce the negative triangle problem to the radius problem. Assume the edge weights in the original graph G are in $\{-n^c, \ldots, n^c\}$. Let $W = 5n^c$.

We will create a graph $\overline{G}$ so that the radius of $\overline{G}$ is less than $3W$ if and only if G has a negative triangle.

Construction Again we create the graph copies I, J, K. This time we create an additional copy I'. We draw edges from I to J, from J to K, and from K to I' if they are present in the original graph, as before. Now, make the following modifications:

(1) Add W to the weight of all of the edges. Now, all edge weights are at least $4n^c = 4W/5$.
(2) For all $i \in I$, add an edge of weight $3W - 1$ to every vertex in $\overline{G}$ except for the copy i' of i in I'.
(3) Add a vertex t with an edge of weight $3W - 1$ to every vertex in I.

Analysis First, we will show that if i, j, k is a negative triangle in G then the radius of $\overline{G}$ is less than $3W$. If there is a negative triangle then $d(i, i') < 3W$ since we added W to all of the edges. Also, there is an edge of weight $3W - 1$ from i to every other vertex in the graph. Thus, i has distance less than $3W$ to every vertex so the radius is less than $3W$.

Now, we will show that if the radius of $\overline{G}$ is less than $3W$ then G has a negative triangle. If the radius of $\overline{G}$ is less than $3W$ then the center must be in I because the distance from t to every vertex in $J \cup K \cup I'$ is at least $3W$. Let $i \in I$ be the center and let i' be the copy of i in I'. Since $d(i, i') < 3W$, the shortest path between i and i' cannot use any of the added edges of weight $3W - 1$ because there is no such edge between i and i'. Thus, any shortest path between i and i' must use only edges from the original graph G. Also, no shortest path between i and i' can have more than 3 edges because each edge has weight at least $4W/5$. Thus, any shortest path between i and i' uses exactly one vertex $j \in J$ and exactly one vertex $k \in K$. This corresponds to a triangle (i, j, k) in G whose weight is negative since $d(i, i') < 3W$ and we added W to the weight of each edge.

□

A graph parameter related to the radius is the *diameter*, which is simply the largest distance in the graph i.e. $\max_{u,v} d(u, v)$.

Big open problem: Is Diameter subcubically equivalent to APSP?