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).

But what if we want to find triangles of *minimum weight*? This would help us find the "least important" three nodes, or the "most important" k-cliques (since triangles can be used to find cliques in graphs). In general, finding triangles of minimum weight would help us find important patterns in graphs.

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. We've already talked about node-weighted APSP, and this problem is equivalent to finding vertices u, v such that $(u, v) \in E$ and the weight of the "smallest" possible 2-path from u to v is minimized.

We can solve this problem using min-witnesses for Boolean matrix multiplication. Specifically, we can compute the product $A \circ B$, where

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

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 $\min_{(i,j)\in E} w(i) + w(j) + w((B \circ B)[i, j])$.

By Lecture 5, this min witness product can be found in $O(n^{2.575})$ time. Therefore, minimum node weight triangles can be found in $O(n^{2.575})$ time.

Later, Czumaj and Lingas proved minimum node weight triangles could be found in $\tilde{O}(n^\omega)$ time.

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.

Proposition 1. *We can reduce the min-edge-weight triangle problem to the* $(\min, +)$ *product.*

Proof. 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 is $\min_{(i,j)\in E} w(i, j) + (A \star A)[i, j]$. □

This is the best known strategy for the min-edge-weight triangle problem! Why? Because the problem is, in some sense, equivalent to APSP.

Theorem 2.1. *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.*

In other words,

$$\text{Min-edge-weight triangle} \equiv_3 \text{APSP}$$

(this notation means that if you have a truly subcubic algorithm for one problem, then you have a truly subcubic algorithm for the other). The rest of this lecture is devoted to proving this.

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 Reductions

We define two additional problems (that are equivalent to our original problem, and to APSP):

All pairs min triangles: Given a weighted tripartite graph on parts I, J, K, find $\min_{v_J \in J} w(u_I, v_J) + w(v_J, t_K) + w(u_I, t_K)$ for all pairs $u_I \in I$, $t_K \in K$.

This problem is equivalent to the distance product (which is equivalent to APSP). The difference between this problem and the problem we are trying to solve is that this problem finds minimum triangle weights for all pairs of vertices, while our original problem finds the smallest triangle in the entire graph. It is easy to reduce the original problem to all-pairs-min-triangles, but to prove equivalence, we must reduce all-pairs-min-triangles to our original problem.

All pairs negative triangles: 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 $w(u_I, v_J) + w(v_J, t_K) + w(u_I, t_K) < 0$.

All pairs negative triangles is easily reducible to all-pairs min triangles (just find the minimum weight for all pairs of vertices, and test if it's less than 0).

So far, we have reduced

- Original problem $\Rightarrow$ All-pairs-min-triangles
- All pairs negative triangles $\Rightarrow$ All-pairs min triangles

Now we go in the other direction.

2.3 Reducing all-pairs min triangles to all pairs negative triangles

Lemma 2.1. *If all pairs negative triangles is in $T(n)$ time, then all pairs min triangles is in $O(T(n) \log M)$ time.*

Proof. For all $u_I \in I$, $t_K \in K$, we can use binary search to guess the value $W_{ut} = \min_{v_J \in J} w(u_I, v_J) + w(v_J, t_K)$. This allows us to guess the value of the minimum weight triangle that uses those vertices.

For each u, t, we guess a value W_{ut}, and replace the edge weight $w(u_I, t_K)$ 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 $\min_{v_J} w(u_I, v_J) + w(v_J, t_K) < -W_{ut}$. Using a simultaneous binary search (for all u, t) over all possible edge weights, we can find the actual value of the minimum weight triangle. This takes $O(T(n) \log M)$ time. □

2.4 Reducing all-pairs negative triangles to our original problem

Suppose we have an algorithm A that detects a negative triangle in $T(n)$ time.

Then pick a parameter L, and partition each of I, J, K into n/L pieces of size L.

Algorithm 2: All-pairs negative triangles (given the ability to find a negative triangle in a graph)

```
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
```

2.4.1 Runtime

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 Applications to graph radius

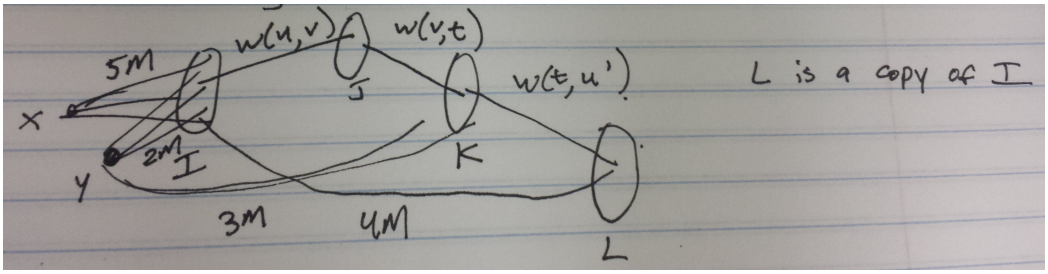
In the **graph radius** problem, we are given an undirected graph with integer edge weights, and want to find

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

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. Below we explain this by showing that the radius problem is subcubically equivalent to APSP.

Theorem 3.1. *Graph radius $\equiv_3$ APSP.*



Proof. Reduce the negative triangle problem to the radius problem. Assume the edge weights in the original graph are in $\{-M, \ldots, M\}$.

Again we create the graph copies I, J, K. This time we create an additional copy L. We draw edges from I to J, from J to K, and from K to L (connecting a vertex in I with a vertex in J if they are connected in the original graph). Add $2M$ to all the edge weights, so now our question is whether there's a path from $u \to v \to t \to u'$ of length $< 6M$.

Now create a separate vertex x with edges to all vertices in I (all these edges have weight $5M$). Then create a vertex y with edges to all vertices in I (all these edges have weight $2M$). y also has edges (of weight $3M$) to all vertices in K. Last but not least, take any node $u \in I$, and any node $v' \in L$ such that $v' \neq u$, and add an edge (u, v') of weight $4M$.

We claim that if $R < 6M$, then

- The center of this graph is in I.
- For all $u \in I$ and $v \in \{x, y\} \cup J \cup K$, we have $d(u, v) \leq 5M$
- For all $u \in I$ and $v \in K$ such that $v \neq u$, we have $d(u, v) \leq 4M$
- For all $u \in I$, we have $d(u, u') = \min\{6M, \text{min weight triangle through} u\}$

So $R < 6M$ if and only if there exists u such that the min weight triangle through u has weight less than $6M$.

□