1 All-Pairs Shortest Paths (APSP)

This lecture will be about All-Pairs Shortest Paths (APSP) and some problems equivalent to it. Let's begin with a definition of APSP:

> **All-Pairs Shortest Paths (APSP):** Given an n-node edge-weighted (directed) graph G with no negative weight cycles, compute for all pairs of nodes i, j the shortest distance between i and j.

The condition requiring no negative weight cycles is necessary for the distances to be well-defined and is standard in shortest paths algorithms.

In this lecture we will focus on the case of APSP where the edge weights are integers, and we will be working in the word-RAM model with $O(\log n)$ bit words. In later lectures we will also consider the case when the weights are real numbers and we will then work in the *Real RAM* model which is an extension of the Word-RAM that we will define in the relevant lectures.

On n-node graphs, the Floyd-Warshall algorithm (from your undergraduate algorithms courses) gives an $O(n^3)$ time algorithm assuming additions and comparisons of weights are constant-time operations. The problem can also be solved in $\tilde{O}(mn + n^2)$ time in m-edge graphs, e.g. via Johnson's algorithm. A major open problem is whether APSP has an $O(n^{3-\varepsilon})$ time algorithm for any constant $\varepsilon > 0$, and the APSP Hypothesis (introduced in lecture 1) asserts that no such algorithm exists.

2 APSP and $(\min, +)$-Product

It has been known for decades that computing APSP is innately related to computing a strange matrix product (sometimes referred to as "funny" matrix multiplication). Formally, the operation is known as $(\min, +)$ matrix product, the matrix product over the so called *tropical semiring*. [1] Given two matrices $A, B \in \mathbb{Z}^{n \times n}$, we define another $n \times n$ matrix

$$(A \star B)[i, j] := \min_k (A[i, k] + B[k, j]).$$

Note, we are substituting min in place of addition, and we are substituting $+$ in place of multiplication.

Theorem 2.1. *Let $T(n) \leq O(n^3)$. APSP on n-node graphs is in $T(O(n))$ time if and only if $(\min, +)$-product on $n \times n$ matrices is in $T(O(n))$ time.*

One direction of the above theorem is pretty straightforward: if APSP has a $T(n)$ time algorithm, then one can use it to solve $(\min, +)$-product via the following natural reduction. Given two $n \times n$ integer matrices A and B, create a *directed* graph G on $3n$ nodes, organized in n-node layers I, J, K, so that the nodes in I correspond to the rows of A, those in J correspond to the columns of A and rows of B and those in K correspond to the columns of B. Then, for all $i \in I, j \in J$, add an edge (i, j) with weight $A[i, j]$ and for all $j \in J, k \in K$ add an edge (j, k) with weight $B[j, k]$. Every path from $i \in I$ to $k \in K$ has the form $i \to j \to k$ for some $j \in J$ and the distance between i and k is exactly

$$d(i, k) = \min_{j \in J} A[i, j] + B[j, k].$$

[1] There is an entire area of "tropical mathematics" where all your favorite objects over $\mathbb{R}$ are studied but addition is replaced by the minimum operation and multiplication is replaced by the sum operation. Interestingly, the algebraic geometers who study these objects think of them as simpler and easier to work with than the usual objects over $\mathbb{R}$ ...

Thus solving APSP on G solves the min-plus product of A and B.

> **Exercise**: How can you make the reduction graph undirected so that the reduction goes through?

Now, let's provide a simple proof of a slightly weaker version of the opposite direction of the theorem.

Lemma 2.1. *Suppose that one can compute the* $(\min,+)-$*Product of two* $n \times n$ *matrices in* $T(n)$ *time, then APSP on* n *node graphs with no negative cycles is in* $O(T(n)\log n)$ *time.*

Proof. Let $G = (V, E)$ be an instance of APSP with weights $w(\cdot,\cdot)$. Define A to be the generalized adjacency matrix. $A[i,j] = w(i,j)$ when $(i,j) \in E$, $w(i,i) = 0$, $w(i,j) = \infty$ if $i \neq j$ and $(i,j) \notin E$.

> **Exercise**: Convince yourself that the ∞ elements above can be replaced by a large enough finite integer. How large does this integer have to be?

Let A^ℓ be $A \star A \star \ldots \star A$, where ℓ copies of A are multiplied. $A^1 = A$.

Claim 1. *For all* i, j, $A^\ell[i,j]$ *is the smallest out of all weights of* i*-*j *paths on at most* ℓ *hops.*

We prove the claim by induction. Clearly, the claim holds for $\ell = 1$, by the definition of A.

Suppose that for some ℓ, for all i, j, $A^\ell[i,j]$ is the smallest out of all weights of i-j paths on at most ℓ hops. Now consider $A^{\ell+1}[a,b]$ for some a, b.

$$A^{\ell+1}[a,b] = \min_{k=1}^{n} A^\ell[a,k] + A[k,b].$$

First notice that by the induction hypothesis, each $A^\ell[a,k]$ is a weight of some path of length at most ℓ from a to k (or ∞ if no such path exists). Thus $A^\ell[a,k] + A[k,b]$ is the weight of some path of length at most $\ell+1$ from a to b (or ∞). Hence, $A^{\ell+1}[a,b]$ is at least the min weight path of length at most $\ell+1$ from a to b. We will also show that it is at most that weight.

Suppose that $P = \{a = a_0 \to a_1 \to \ldots \to a_t = b\}$ be a shortest $a-b$ path among those on $\leq \ell+1$ hops. If $t \leq \ell$, then $A^{\ell+1}[a,b] \leq A^\ell[a,b] + A[b,b] = A^\ell[a,b]$ which is the smallest weight of a path on at most ℓ hops by induction, and is thus $= w(P)$. If on the other hand, $t = \ell+1$, then

$$A^{\ell+1}[a,b] = \min_{k=1}^{n} A^\ell[a,k] + A[k,b] \leq A^\ell[a,a_\ell] + A[a_\ell,b].$$

Since the portion P' of P from a to a_ℓ must be a min weight path among those of length $\leq \ell$ (as otherwise P would not be shortest), $A^\ell[a,\ell] = w(P')$, and so $A^{\ell+1}[a,b] = w(P') + w(a_\ell,b) = w(P)$. **[end of proof of claim]**

Claim 2. *If a graph does not have negative weight cycles, then for any pair of vertices* u *and* v *s.t.* u *can reach* v*, there is a* u *to* v *shortest path that is simple, i.e. it does not have any repeated vertices. (This is known as "Shortest Paths are, without loss of generality, simple.")*

> **Exercise**: Prove the above claim.

Because the shortest paths we care about are simple, they have at most $n-1$ hops. This means that to compute the distances in G, it suffices to compute A^{n-1}, or any power A^p with $p \geq n-1$.

We can do this via successive squaring: Assume that we have computed A^{2^j} for some j, then we can compute $A^{2^{j+1}} = A^{2^j} \star A^{2^j}$ via a single product. We start from $A = A^{2^0}$ and using $\lceil \log_2(n-1) \rceil$ products (successive squarings), we can compute A^p with $p \geq n-1$.

Thus, if $(\min,+)$-product is in $T(n)$ time, then APSP is in $O(T(n)\log n)$ time. □

In a future problem set, you will prove that with a mild condition on $T(n)$, the log factor can be removed. This condition holds for most running time functions that we care about, and hence APSP and $(\min, +)$-product are runtime-equivalent, within constant factors.

3 Negative and Minimum Triangles

Suppose we have an n node graph with *edge* weights $w : E \to \mathbb{Z}$. The *Min-Weight Triangle* problem is to find vertices i, j, k minimizing $w(i, j) + w(j, k) + w(i, k)$. There is **no 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 triples of vertices.

A similar problem is the *Negative Triangle* Problem in which one is given a graph with integer edge weights, and one needs to decide whether there exist three nodes i, j, k with $w(i, j) + w(j, k) + w(i, k) < 0$. Clearly, if one can find a Min-Weight Triangle in $T(n)$ time, then one can check if its weight is negative and can thus also detect a Negative Triangle.

Proposition 1. *We can reduce the Min-Weight Triangle problem on n node graphs, in $O(n^2)$ time to the $(\min, +)$ product of $n \times n$ matrices.*

> **Exercise:** Prove the above Proposition.

This is the best known strategy for the Min-Weight triangle problem! Why? Because the problem is, in some sense, equivalent to APSP (which is equivalent to $(\min, +)$ product). We show below that APSP can even be reduced to Negative Triangle, thus showing that Negative Triangle, Min-Weight Triangle and APSP are "subcubically equivalent": if one of the problems can be solved in $O(n^{3-\varepsilon})$ time for some $\varepsilon > 0$, then all of them can be solved in $O(n^{3-\varepsilon'})$ time for some $\varepsilon' > 0$. This latter running time is called truly subcubic.

Theorem 3.1. *If for some $\epsilon > 0$, the Negative Triangle Problem can be solved in $O(n^{3-\epsilon})$ time, then APSP in n node graphs with edge weights in $\{-W, \ldots, W\}$ and no negative cycles is in $\tilde{O}(n^{3-\epsilon/3} \log(Wn))$ time.*

In other words,

$$\text{Negative Triangle} \equiv_3 \text{APSP}$$

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

3.1 Preliminaries

Without loss of generality, we can assume that for a Negative Triangle Instance:

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 the weight of this triangle is $\geq 6M - 2M = 4M > 3M$, i.e. greater than the weight of any real triangle.
2. G is tripartite.

> **Exercise:** Convince yourself of point 2 above. (This should be similar to some of your proofs on the problem set.)

3.2 Reductions

We define two intermediate problems:

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 $u_I \in I$, $t_K \in K$.

It is not hard to see that this problem is equivalent to the $(min, +)$-product (which is equivalent to APSP).

All Pairs Negative Triangles (APNT): 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 $w(u_I, v_J) + w(v_J, t_K) + w(u_I, t_K) < 0$.

APNT 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), but we would like to reduce All-Pairs Min Triangles (and thus APSP) to APNT. APNT also easily solves Negative Triangle, but we would like to reduce it to Negative Triangle.

3.3 Reducing All-Pairs Min Triangles (and thus APSP) to APNT

Lemma 3.1. *If APNT is in $T(n)$ time, then All Pairs Min Triangles is in $O(T(n) \log M)$ time (where the edge weights of the All Pairs Min Triangles instance are in $\{-M, \ldots, M\}$).*

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. □

3.4 Reducing APNT to Negative Triangle

We first claim that finding can be efficiently reduced to detection:

Claim 3. *Suppose we have an algorithm A that detects a negative triangle in $T(n) = O(n^{3-\varepsilon})$ time for $\varepsilon > 0$. Then we also have an algorithm that can find a negative triangle (if one exists) in $O(n^{3-\varepsilon})$ time.*

> **Exercise:** Prove the above claim.
>
> *Hint:* Split the vertices into roughly equal parts and find a way to recurse.

Now that we have that a negative triangle detection algorithm can be used to find a negative triangle, we can assume that we are given an $O(n^{3-\varepsilon})$ time for $\varepsilon > 0$ algorithm for finding a negative triangle, if one exists.

In Algorithm 1, we give an efficient reduction from APNT to Negative Triangle (NT) finding. Combined with the finding to detection reduction, we obtain a reduction from APNT to Negative Triangle detection.

> **Exercise:** Convince yourself that the Algorithm is correct, i.e. for every $a \in I, c \in K$, $C[a, c] = 1$ if and only if there is some $b \in J$ such that a, b, c is a negative triangle in G.

3.4.1 Runtime

This algorithm runs in time

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

Algorithm 1: All-pairs negative triangles (given the ability to find a negative triangle in a graph)

Begin APNT to NT reduction:
We are given $G = (I \cup J \cup K, E)$, tripartite weighted graph.
Partition I, J, K into $\{I_1, \ldots, I_{n/L}\}, \{J_1, \ldots, J_{n/L}\}, \{K_1, \ldots, K_{n/L}\}$.
Initialize C to an $n \times n$ 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 triples* (i,j,k)*, where* i,j,k *range from* 1 *to* n/L **do**
  Consider G_{ijk}, the subgraph of G induced by $I_i \cup J_j \cup K_k$.
  while G_{ijk} *contains a negative triangle (⋆ A call to NT algorithm ⋆)* **do**
    Let a_I, b_J, c_K be the nodes of the triangle returned by the NT alg.
    Set $C[a_I, c_K] = 1$.
    Delete (a_I, c_K) from G (this deletes it from all the induced subgraphs G_{ijk}).
return C
End APNT to NT reduction

The runtime is dominated by the number of times a call to Negative Triangle finding happens (and each such call takes $T(L)$ time). There are two types of such calls. The first type are those that return a negative triangle. The total number of such calls is no more than n^2 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).

The second type of calls to Negative Triangle are those that do not find a negative triangle. The total number of such calls is exactly one for each triple (i,j,k), making sure that G_{ijk} has no more negative triangles. Thus the number of such calls is the $(n/L)^3$ term.

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 $O(n^{3-\epsilon/3})$.

3.4.2 Notes

The reduction from APNT to NT didn't use anything about the fact that the triangle property we use is that it is negative. It is a generic reduction from the All-Pairs version of any triangle detection problem to the single triangle detection version. Thus, in particular we get that

> **Boolean Matrix Multiplication** ("BMM") (which is equivalent to All-Pairs triangle detection) can be (n^3, n^3) reduced to **Triangle Detection.** Since BMM can be used to solve Triangle Detection, we get that these two problems are (n^3, n^3)-**equivalent**.

Since the reductions in both directions are simple and combinatorial, we get that any simple and combinatorial algorithm for Triangle Detection running in truly subcubic time also implies such an algorithm for BMM, and vice versa.

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

Theorem 4.1. *Graph Radius* $\equiv_3$ *APSP.*

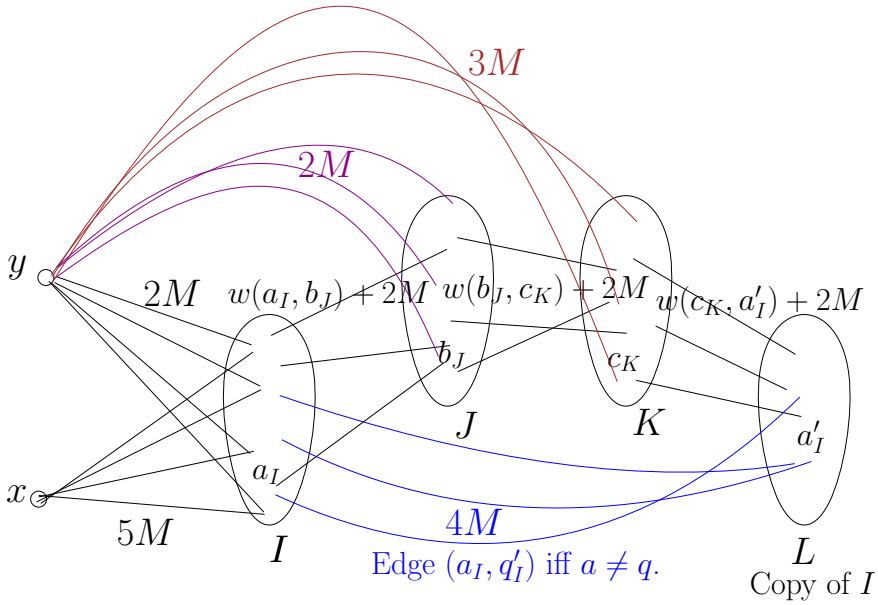

Figure 1: Reduction from Negative Triangle to Graph Radius. The weights $w(e)$ for edges in $I \times J, J \times K, K \times L$ are $2M + w_G(e)$, where $w_G(e)$ is the weight in G of the edge corresponding to e in G.

Proof. Reduce the negative triangle problem to the radius problem. We can assume that we are given a tripartite graph $G = (V, E)$ where the three vertex partitions are I, J, K and the edge weights in G are integers in $\{-M, \ldots, M\}$ for some integer M.

We will create a graph H which will be an instance of the Radius problem. The vertices of H will consist of I, J, K (corresponding directly to the vertices in G) and one more set of vertices L which will be a copy of I. That is, each node $u \in I$ has a copy $u' \in L$. We add two more additional vertices x and y.

We draw edges from I to J, from J to K, and from K to L. The edge sets between I and J and between J and K are the same as those in G. The edges from K to L are the same as those between K and I in G (recall L is a copy of I). So far every edge in H is in direct correspondence with an edge in G. The weight of an edge in H is $2M+$ the weight of the corresponding edge in G. In particular this makes all edge weights in the graph $\geq M$.

The proof of the claim below is simple:

Claim 4. *A node $u \in I$ appears in a negative triangle in G if and only if there is a path from $u \in I$ to $u' \in L$ in H of weight $< 6M$. (Recall that u' is the copy of $u \in I$ in L.)*

Now we add edges between the special new vertices x and y and the rest of H. We add edges from x to all vertices in I (all these edges have weight $5M$). Then we add edges from y to all vertices in I (all these edges have weight $2M$). Node y also has edges (of weight $3M$) to all vertices in K and edges (of weight $2M$) to all vertices in J. 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 $4M$.

The construction is depicted in Figure 1.

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

- The center of this graph is in I.

> **Exercise:** Show that this is the case, i.e. that every node not in I is at distance $\geq 6M$ from some other node.

- 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 $v' \in L$ such that $v \neq u$, we have $d(u, v') \leq 4M$.
- For all $u \in I$, we have $d(u, u') = 6M + \min\{0, \text{min weight of a triangle through } u\}$.

Exercise: Verify the last three claim bullets above.

So $R < 6M$ if and only if there exists $u \in I$ such that the min weight triangle through u has weight less than 0. Thus $R < 6M$ if and only if the original Negative Triangle instance graph contains a negative triangle. □